aws

User Guide

Amazon CloudWatch



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon CloudWatch: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon CloudWatch? 1
Monitor applications and AI agents with CloudWatch Omni 1
Operational visibility with metrics, alarms, and dashboards 2
Application performance monitoring (APM) 2
Infrastructure monitoring 3
Collect, store, and query logs 3
Use the CloudWatch agent to gather metrics, logs, and traces from Amazon EC2 fleets 3
Cross-account monitoring 4
OpenTelemetry support 4
Solutions catalog 4
CloudWatch network monitoring 4
Billing and costs 5
Resources 5
Quickstart 7
Choosing the right path 7
Prerequisites 8
Step 1: Instrumenting your application 10
Step 2: Running the Collector 12
Step 3: Checking that your telemetry arrived 14
Step 4: Creating an alarm and a dashboard widget 16
Troubleshooting telemetry export 17
Clean up resources 19
Keeping costs predictable 19
Next steps 20
Collect and send telemetry 21
OpenTelemetry 21
OTLP Endpoints 23
Getting started 34
Bearer token authentication for Metrics 73
Troubleshooting 85
CloudWatch agent 89
Supported operating systems 92
Prerequisites 93
Download the CloudWatch agent package 97

Verifying the signature of the CloudWatch agent package ... 103
Installing the CloudWatch agent ... 113
Set up the CloudWatch agent with security-enhanced Linux (SELinux) ... 154
Create the CloudWatch agent configuration file ... 157
Start the CloudWatch agent ... 332
Metrics collected by the CloudWatch agent ... 335
Using the CloudWatch agent with related telemetry ... 354
Common scenarios with the CloudWatch agent ... 355
CloudWatch agent credentials preference ... 365
Troubleshooting the CloudWatch agent ... 367
Embedding metrics within logs ... 377
Publishing logs with the embedded metric format ... 379
Viewing your metrics and logs in the console ... 402
Setting alarms on metrics created with the embedded metric format ... 404
CloudWatch pipelines ... 404
Pipeline components ... 405
Pricing ... 406
Region availability ... 406
Creating pipelines ... 408
Managing pipelines ... 412
Supported data sources for CloudWatch ... 412
Processors ... 739
Expression syntax for conditional processing ... 785
Sinks ... 790
CloudWatch pipelines IAM policies and permissions ... 795
Extensions ... 820
Monitoring pipelines ... 822
Troubleshooting ... 829
Managed Prometheus collectors ... 830
Integrate Amazon MSK ... 831
Integrate Amazon OpenSearch Service ... 842
Integrate Amazon EKS ... 845
VPC-connected managed collector ... 847
Scraper configuration ... 851
Monitor collectors ... 853
CloudWatch Omni ... 855

How CloudWatch Omni is organized ... 855
What you can do in CloudWatch Omni ... 856
How Omni relates to CloudWatch ... 857
Get started ... 857
Set up Omni ... 858
Choose your setup path ... 858
After setup ... 860
Set up Omni for a single account ... 861
Set up Omni for your organization ... 865
Control access to your space ... 871
AWS managed policies for CloudWatch Omni ... 904
Send telemetry to CloudWatch Omni ... 908
Where to start ... 908
How telemetry reaches your space ... 909
Forwarding to the Dataset ... 910
Choose how you collect ... 912
Choose an endpoint ... 912
Authentication ... 913
Environment support ... 913
Next steps ... 913
Send application telemetry ... 914
Send AI agent telemetry ... 946
Supported environments, languages, and frameworks ... 967
Ask the Omni agent ... 975
Where you can ask ... 976
What you can ask ... 976
How the agent works with you ... 977
Limitations ... 978
Permissions and data access ... 978
Investigations with the AWS DevOps Agent ... 978
Investigate with the AWS DevOps Agent ... 978
Explore and query your telemetry ... 980
Run your first query ... 980
Query logs and traces with SQL ... 981
Query metrics with PromQL ... 982
Discover fields ... 982

Work with results ... 983
CloudWatch Omni SQL limits ... 983
Act on what you find ... 983
SQL query reference ... 983
Dashboards ... 988
Every dashboard is shared ... 989
Pages and dashboards ... 989
What goes on a dashboard ... 989
Create a dashboard ... 989
Manage your dashboards ... 990
Threads ... 990
What a thread holds ... 990
Start a thread ... 991
Manage your threads ... 991
Collaborate and share ... 991
Alerts ... 991
Alert query ... 992
Thresholds and alert states ... 992
Evaluation timing ... 992
One alert, many series ... 993
Create an alert ... 993
Send notifications ... 994
Manage your alerts ... 994
Investigate a firing alert ... 994
Monitor AI agents ... 995
What you can do ... 996
How an agent run is represented ... 996
Where to start ... 997
Available across two surfaces ... 997
Feature availability by surface ... 998
Where your agent runs ... 998
How Omni relates to AgentCore and generative AI observability ... 999
Protect sensitive data ... 999
Analyze agent behavior ... 1004
Agent traces ... 1005
Agent sessions ... 1009

Agent topology ... 1011
Evaluate agent quality ... 1013
Evaluators and evaluations ... 1015
Datasets ... 1019
Experiments ... 1022
Online evaluations ... 1023
Prompt playground ... 1025
Monitor agents in the web UI ... 1027
Develop with the IDE extension ... 1028
Manage prompts ... 1032
Monitor applications with the application map ... 1034
Find which services need your attention ... 1034
Understand what appears on the map ... 1035
Investigate an application or a service from the map ... 1036
Narrow and organize the map ... 1037
What to watch out for ... 1037
Where to go next ... 1038
Cross-account and cross-Region topology ... 1038
Using Context Graph ... 1039
Monitor a service ... 1055
Application traces ... 1058
Connect Slack ... 1063
Disconnect an integration ... 1066
Who can manage integrations ... 1066
Use Omni skills ... 1066
How skills work with your AWS credentials ... 1067
What your coding agent can do with Omni ... 1067
Install the Omni skills ... 1067
To install and use the Omni skills ... 1068
Security in CloudWatch Omni ... 1068
Data encryption in CloudWatch Omni ... 1069
Identity and access management for CloudWatch Omni ... 1077
Compliance validation for CloudWatch Omni ... 1086
Resilience in CloudWatch Omni ... 1086
Infrastructure security in CloudWatch Omni ... 1087
Interface VPC endpoints (AWS PrivateLink) ... 1088

Configuration and vulnerability analysis in CloudWatch Omni ... 1091
Cross-service confused deputy prevention ... 1092
Security best practices for CloudWatch Omni ... 1094
Opting out of using your data for service improvement ... 1096
IAM policies to use CloudWatch Omni ... 1099
Cross-Region inference in CloudWatch Omni ... 1108
Tutorial: Monitor an application ... 1110
Prerequisites ... 1110
Step 1: Instrument your application ... 1111
Step 2: Send telemetry to CloudWatch ... 1111
Step 3: See your application on the map ... 1112
Step 4: Examine service health ... 1112
Next steps ... 1112
Tutorial: Fix a production quality regression ... 1113
What you will do ... 1113
Prerequisites ... 1114
Choose your agent ... 1115
Phase A: Build and instrument (IDE extension) ... 1117
Phase B: Deploy and observe (Omni web UI) ... 1119
Phase C: Reproduce and fix (IDE extension) ... 1122
Phase D: Verify continuously (Omni web UI) ... 1127
What you keep ... 1127
Dashboards ... 1129
Getting started with automatic dashboards ... 1131
Viewing the cross-service dashboard ... 1132
Removing a service from appearing in the cross-service dashboard ... 1134
Viewing a dashboard for a single service ... 1134
Viewing a dashboard for a resource group ... 1136
Creating a customized dashboard ... 1137
Creating a cross-account cross-Region dashboard with the console ... 1138
Create a cross-account cross-Region dashboard programmatically ... 1139
Creating a graph with metrics from different accounts and Regions ... 1143
Adding an alarm from a different account to a cross-account dashboard ... 1143
Creating dashboards with variables ... 1144
Types of dashboard variables ... 1144
Copying a variable to another dashboard ... 1145

Tutorial: Using a regular expression pattern to switch between Regions 1146
Tutorial: Creating dashboard with the Lambda function name as a variable 1147
Using widgets on dashboards 1149
Adding a graph widget 1151
Removing a graph widget 1154
Graph metrics manually on a dashboard 1154
Editing a graph 1156
Renaming a graph 1162
Moving a graph 1163
Changing the size of graphs 1163
Changing the size of a graph temporarily 1163
Add a metrics explorer widget 1164
Adding a line graph widget 1165
Removing a line graph 1166
Adding a number widget 1166
Removing a number widget 1167
Adding a gauge widget 1167
Removing a gauge widget 1168
Using a custom widget 1168
Adding a text widget 1179
Editing a text widget 1180
Removing a text widget 1180
Adding an alarm 1180
Adding an alarm status widget 1181
Removing an alert widget 1181
Using a data table widget 1181
Linking graphs 1185
Unlinking graphs 1185
Sharing dashboards 1186
Permissions required to share a dashboard 1187
Permissions that are granted to people who you share the dashboard with 1189
Sharing one dashboard with specific users 1190
Sharing a dashboard publicly 1191
Sharing all dashboards using SSO 1192
Setting up SSO 1193
Seeing how many of your dashboards are shared 1194

Seeing which dashboards are shared ... 1194
Stopping dashboard sharing ... 1195
Reviewing and changing shared dashboard permissions ... 1195
Allowing people that you share with to see composite alarms ... 1197
Allow widgets that use PromQL queries to be visible in shared dashboards ... 1198
Allowing people that you share with to see logs table widgets ... 1199
Allowing people that you share with to see custom widgets ... 1200
Using live data ... 1202
Using live data for a dashboard ... 1202
Using live data for a widget ... 1203
Viewing an animated dashboard ... 1203
Add a dashboard to your favorites list ... 1204
Changing the period override setting or refresh interval ... 1204
Changing the time range or time zone format ... 1206
Dashboard Body Structure and Syntax ... 1208
Overall Structure ... 1209
Widgets Array Structure ... 1214
Variables Array Structure ... 1216
Properties of a Log Widget Object ... 1223
Properties of a Metric Widget Object ... 1225
Metric Widget: Format for Each Metric in the Array ... 1231
Properties of a Metrics Explorer Widget Object ... 1246
Properties of an Alarm Status Widget Object ... 1253
Properties of a Chart Widget Object ... 1255
Graph metrics ... 1267
Graph a metric ... 1267
Merge two graphs into one ... 1273
Use dynamic labels ... 1274
Modify the time range or time zone format for a graph ... 1277
Zooming in on a graph ... 1280
Modify the y-axis for a graph ... 1282
Create an alarm from a metric on a graph ... 1283
GetMetricWidgetImage: Metric Widget Structure and Syntax ... 1284
Explore related telemetry ... 1301
What is related telemetry? ... 1302
How to access ... 1302

Navigating related telemetry ... 1303
Using the topology map ... 1306
Finding a specific resource ... 1306
Prerequisites ... 1307
How does CloudWatch find related telemetry? ... 1308
AWS services that support related telemetry ... 1310
Add related information to custom telemetry ... 1324
Metrics ... 1331
Overview ... 1331
OpenTelemetry: open-source native metrics in CloudWatch ... 1331
Two metric models ... 1331
Getting started ... 1332
OpenTelemetry Metrics (Recommended) ... 1332
Overview ... 1333
Publish custom metrics ... 1334
Histogram metrics ... 1338
Query metrics with PromQL ... 1340
Anomaly detection using PromQL ... 1373
Detailed monitoring ... 1374
Pricing and storage ... 1379
Migrate from Classic ... 1381
Vended Metrics in PromQL ... 1383
Resource tags for telemetry ... 1741
Enable resource tags on telemetry ... 1742
Using resource tags for telemetry ... 1743
Disable resource tags on telemetry ... 1937
Troubleshooting ... 1938
CloudWatch Metrics (Classic) ... 1939
Concepts ... 1940
Basic and detailed monitoring ... 1949
Publish custom metrics ... 1951
Query metrics with CloudWatch Metrics Insights ... 1956
View available metrics ... 1992
GetMetricData (Primary API) ... 1997
GetMetricStatistics (Legacy) ... 1999
Use metrics explorer to monitor resources by their tags and properties ... 2004

Search expressions ... 2007
Use metric streams ... 2019
Math expressions ... 2067
Using anomaly detection ... 2114
Query metrics from other data sources ... 2118
Grafana integration ... 2137
Services that publish CloudWatch metrics ... 2137
Alarms ... 2154
Concepts ... 2156
Alarm data queries ... 2156
Alarm evaluation ... 2159
PromQL alarms ... 2177
Log alarms ... 2178
Composite alarms ... 2189
Alarm actions ... 2200
Alarm Mute Rules ... 2205
Limits ... 2217
Getting started ... 2221
Create alarms ... 2221
Acting on alarm changes ... 2261
Configure alarm mute rules ... 2315
Managing alarms ... 2317
Editing or deleting a CloudWatch alarm ... 2318
Hide Auto Scaling alarms ... 2319
Alarms and tagging ... 2320
Viewing and managing muted alarms ... 2320
Common features of CloudWatch alarms ... 2322
Alarm recommendations for AWS services ... 2323
Find and create recommended alarms ... 2324
Recommended PromQL alarms ... 2326
Recommended alarms (Classic) ... 2399
Alarm use cases and examples ... 2503
Create a billing alarm ... 2503
Create a CPU usage alarm ... 2507
Create a load balancer latency alarm ... 2510
Create a storage throughput alarm ... 2512

Create an alarm on Performance Insights counter metrics from an AWS database 2514
Create a Log Alarm on errors 2516
Create a Log Alarm on latency 2517
Create an Auto Scaling action alarm 2518
Telemetry discovery and enablement 2521
What is telemetry discovery and enablement? 2521
Setting up telemetry configuration 2523
Prerequisites and permissions 2524
Enable telemetry configuration for your account 2526
Enable telemetry configuration for your organization 2527
Registering a delegated administrator account for your organization 2529
Discovering resources 2531
Viewing data sources 2532
Viewing discovered resources 2532
Filtering and preferences 2533
Telemetry enablement rules 2535
How rules work 2535
Creating a telemetry enablement rule 2538
Managing telemetry rules 2539
Encrypting log groups with customer managed keys 2540
Supported data sources 2543
Troubleshooting telemetry configuration 2549
Resources not appearing 2549
Rules not applying 2549
Multi-Region issues 2550
Home Region conflicts 2551
Disabling telemetry configuration 2551
Infrastructure monitoring 2552
Container Insights 2552
Amazon EKS 2554
Amazon ECS 2681
Setting up Container Insights on Red Hat OpenShift on AWS (ROSA) 2701
Viewing Container Insights metrics 2702
Metrics collected by Container Insights 2706
Lambda Insights 3020
Get started with Lambda Insights 3021

Viewing your Lambda Insights metrics ... 3117
Integration with Application Insights ... 3119
Metrics collected by Lambda Insights ... 3119
Troubleshooting and known issues ... 3132
Example telemetry event ... 3133
Database Insights ... 3138
Modes ... 3139
Data retention ... 3141
Pricing ... 3142
Get started ... 3142
Monitoring Self-Managed Databases ... 3146
Cross-account cross-region monitoring ... 3152
Viewing the Fleet Health Dashboard ... 3155
Viewing the Database Instance Dashboard ... 3163
Monitoring Aurora Limitless databases with Database Insights ... 3188
Troubleshooting ... 3196
SageMaker AI Insights ... 3198
Key capabilities ... 3198
Architecture and data flow ... 3199
How to access ... 3199
Prerequisites ... 3200
Pricing ... 3200
Get started ... 3200
Insights dashboard ... 3203
Metrics reference ... 3211
Troubleshooting ... 3216
ElastiCache Insights ... 3219
Key capabilities ... 3219
Supported configurations ... 3219
How to access ... 3219
Pricing ... 3220
Get started ... 3220
Use Contributor Insights to analyze high-cardinality data ... 3221
Create a Contributor Insights rule in CloudWatch ... 3222
Contributor Insights rule syntax in CloudWatch ... 3228
Example rules ... 3232

Viewing Contributor Insights reports in CloudWatch ... 3236
Graphing metrics generated by rules in CloudWatch ... 3237
Using Contributor Insights built-in rules in CloudWatch ... 3240
CloudWatch Application Insights ... 3241
What is Amazon CloudWatch Application Insights? ... 3242
How Application Insights works ... 3252
Prerequisites, IAM policies, and permissions ... 3268
Set up application for monitoring ... 3272
Application Insights cross-account observability ... 3301
Work with component configurations ... 3302
Use CloudFormation templates ... 3374
Tutorial: Set up monitoring for SAP ASE ... 3387
Tutorial: Set up monitoring for SAP HANA ... 3396
Tutorial: Set up monitoring for SAP NetWeaver ... 3412
View and troubleshoot Application Insights ... 3429
Supported logs and metrics ... 3433
Using the resource health view ... 3530
Prerequisites ... 3530
Application performance monitoring (APM) ... 3533
Application Signals ... 3534
Features ... 3535
Permissions required for Application Signals ... 3540
Supported systems ... 3544
Supported instrumentation setups ... 3551
Enable Application Signals in your account ... 3552
(Optional) Try out Application Signals with a sample app ... 3553
Enable your applications on Amazon EKS clusters ... 3557
Enable your applications on Amazon EC2 ... 3571
Enable your applications on Amazon ECS ... 3588
Enable your applications on Kubernetes ... 3676
Enable your applications on Lambda ... 3684
Monitor service events ... 3708
Debug applications with Dynamic Instrumentation ... 3716
Troubleshooting your Application Signals installation ... 3723
(Optional) Configuring Application Signals ... 3740
Monitor your application's operational health ... 3753

Metrics collected by Application Signals ... 3811
Custom metrics with Application Signals ... 3826
Service level objectives (SLOs) ... 3833
SLO concepts ... 3834
Calculate error budget and attainment for period-based SLOs ... 3835
Calculate error budget and attainment for request-based SLOs ... 3837
Calculate burn rates and optionally set burn rate alarms ... 3838
Create an SLO ... 3843
Use SLO recommendations ... 3855
View and triage SLO status ... 3856
Edit an existing SLO ... 3858
Delete an SLO ... 3858
Transaction Search ... 3859
Benefits ... 3860
How it works ... 3861
Pricing ... 3862
Enable transaction search ... 3862
Spans ... 3877
Adding custom attributes ... 3888
Troubleshooting application issues ... 3891
Synthetic monitoring (canaries) ... 3893
Required roles and permissions ... 3895
Creating a canary ... 3913
Multilocation canaries ... 4109
Groups ... 4118
Test a canary locally ... 4119
Troubleshooting a failed canary ... 4400
Sample code for canary scripts ... 4410
Canaries and X-Ray tracing ... 4418
Running a canary on a VPC ... 4419
Encrypting canary artifacts ... 4420
Viewing canary statistics and details ... 4423
CloudWatch metrics published by canaries ... 4425
Edit or delete a canary ... 4429
Start, stop, delete, or update runtime for multiple canaries ... 4432
Monitoring canary events with Amazon EventBridge ... 4432

Performing safe canary updates ... 4437
CloudWatch RUM ... 4441
Set up a mobile application to use CloudWatch RUM ... 4444
IAM policies to use CloudWatch RUM ... 4446
Set up a web application to use CloudWatch RUM ... 4449
Using resource-based policies with CloudWatch RUM ... 4459
Configuring the CloudWatch RUM web client ... 4461
Session replay ... 4463
Enabling unminification of JavaScript error stack traces ... 4465
Regionalization ... 4472
Use page groups ... 4473
Specify custom metadata ... 4473
Send custom events ... 4479
Viewing the CloudWatch RUM dashboard ... 4482
CloudWatch metrics that you can collect with CloudWatch RUM ... 4498
Data protection and data privacy with CloudWatch RUM ... 4515
Information collected by the CloudWatch RUM web client ... 4516
Manage your applications that use CloudWatch RUM ... 4550
Troubleshooting ... 4552
Generative AI observability ... 4553
Model Invocations ... 4554
Enabling model invocation in Amazon Bedrock ... 4554
Amazon Bedrock AgentCore ... 4557
Getting started ... 4558
Agents ... 4562
Memory ... 4578
Built-in tools ... 4582
Gateways ... 4587
Identity ... 4592
Coding Agent Insights ... 4596
Prerequisites ... 4596
Setup guides ... 4596
View the dashboards ... 4597
Claude Code ... 4597
OpenAI Codex ... 4605
GitHub Copilot ... 4610

CloudWatch investigations 4614
Methods to create an investigation 4617
Understanding hypothesis visualizations 4617
How CloudWatch investigations finds data for suggestions 4618
Costs associated with CloudWatch investigations 4619
Insights that CloudWatch investigations can surface in investigations 4620
AWS services where investigations are supported 4620
Conduct an CloudWatch investigation without additional configuration 4622
To start an investigation 4622
Configure CloudWatch investigations 4623
See a sample investigation 4623
Set up an investigation group 4624
Configure alarms to create investigations 4628
Integration with Amazon EKS 4634
(Recommended) Best practices to enhance investigations 4637
CloudWatch agent 4638
AWS CloudTrail 4638
CloudWatch Application Signals 4639
X-Ray 4639
Container insights 4639
Database insights 4639
Investigate operational issues in your environment 4640
Create an investigation 4640
Create an investigation from a CloudWatch Application Signals Service Level Objective (SLO) 4641
View and continue an open investigation 4642
Reviewing and executing suggested runbook remediations 4645
Manage your current investigations 4649
Restart an archived investigation 4649
Cross-account investigations 4650
Prerequisites 4651
Setup your monitoring account for cross-account access 4651
Setup your source account(s) for cross-account access 4654
Investigating multi-account issues 4655
Generate incident reports 4655
Understanding AI-derived facts in incident reports 4658

Incident report terminology ... 4660
Generate a report from an investigation ... 4661
Using 5 Whys analysis in incident reports ... 4664
Integrations with other systems ... 4670
Integration with AWS Systems Manager Automation ... 4671
Integration with third-party chat systems ... 4671
Security in CloudWatch investigations ... 4673
Default CloudWatch investigations permissions, retention, and encryption ... 4674
User permissions for your CloudWatch investigations group ... 4674
Additional permissions for Database Insights ... 4676
How to control what data CloudWatch investigations has access to during investigations 4676
Encryption of investigation data ... 4678
Cross-Region inference ... 4685
CloudWatch investigations data retention ... 4687
Incident report retention ... 4687
Troubleshooting ... 4687
CloudWatch investigations cannot assume the necessary IAM roles or permissions. Please verify required roles and permissions are correctly configured ... 4688
Unable to identify event source. Verify that the resource exists in your application topology and the resource type is supported. ... 4688
Analysis complete. Submit additional findings to receive updated suggestions ... 4688
Source account status shows "Pending link to monitoring account" ... 4689
Incident report generation issues ... 4689
Network Monitoring ... 4690
Using Network Flow Monitor ... 4691
What is Network Flow Monitor? ... 4691
Get started ... 4698
API operations ... 4699
Examples with the CLI ... 4701
Work with EKS ... 4714
Install EC2 agents ... 4729
Initialize Network Flow Monitor ... 4749
Monitor network flows ... 4758
View CloudWatch metrics ... 4768
Create alarms ... 4769
CloudTrail ... 4771

Troubleshoot issues ... 4771
Security ... 4772
Using Internet Monitor ... 4783
What is Internet Monitor? ... 4784
Getting started ... 4803
Configure a monitor ... 4806
Examples with the CLI ... 4821
Internet Monitor dashboard ... 4831
Explore data using tools ... 4843
Add a monitor with another AWS service ... 4865
Create alarms ... 4871
EventBridge integration ... 4873
Logs and metrics access errors ... 4874
Data protection and data privacy ... 4875
Identity and Access Management ... 4876
Using Network Synthetic Monitor ... 4895
Key features ... 4895
Terminology and components ... 4896
Requirements and limitations ... 4897
How Network Synthetic Monitor works ... 4897
Supported AWS Regions ... 4901
Pricing ... 4903
API operations ... 4903
Working with monitors and probes ... 4904
Network Synthetic Monitor dashboards ... 4916
Probe alarms ... 4923
Security ... 4923
Identify and access management ... 4925
CloudWatch observability solutions ... 4943
JVM workload on EC2 ... 4945
Requirements ... 4946
Benefits ... 4946
Costs ... 4948
CloudWatch agent configuration for this solution ... 4949
Deploy the agent for your solution ... 4953
Create the JVM solution dashboard ... 4957

NGINX workload on EC2 4961
Requirements 4962
Benefits 4962
Costs 4963
CloudWatch agent configuration for this solution 4964
Deploy the agent for your solution 4970
Create the NGINX solution dashboard 4975
NVIDIA GPU workload on EC2 4977
Requirements 4978
Benefits 4978
CloudWatch agent configuration for this solution 4981
Deploy the agent for your solution 4983
Create the NVIDIA GPU solution dashboard 4987
Kafka workload on EC2 4990
Requirements 4991
Benefits 4991
Costs 4994
CloudWatch agent configuration for this solution 4995
Deploy the agent for your solution 5002
Create the Kafka solution dashboard 5006
Configure the agent for multiple Kafka roles on the same instance 5010
Tomcat workload on EC2 5012
Requirements 5012
Benefits 5013
Costs 5016
CloudWatch agent configuration for this solution 5017
Deploy the agent for your solution 5021
Create the Tomcat solution dashboard 5025
EC2 health 5029
Requirements 5029
Benefits 5030
Costs 5032
CloudWatch agent configuration for this solution 5033
Deploy the agent for your solution 5035
Create the EC2 Health solution dashboard 5039
Get started with the EC2 Health solution dashboard 5041

Prometheus metric collection on Amazon Amazon EC2 ... 5042
Requirements ... 5043
Enable Prometheus exporters ... 5043
Step 1: Set up the scrape configuration ... 5044
Step 2: Set up the scraper ... 5045
Validate metrics collection ... 5046
Build a custom dashboard ... 5047
Costs ... 5047
Prometheus metric collection on Amazon ECS ... 5047
Requirements ... 5048
Enable Prometheus exporters ... 5048
Step 1: Set up security groups ... 5049
Step 2: Set up the scrape configuration ... 5049
Step 3: Set up the scraper ... 5050
Validate metrics collection ... 5051
Build a custom dashboard ... 5051
Costs ... 5051
Amazon MSK insights ... 5051
Requirements ... 5052
Enable the managed collector ... 5052
Amazon MSK detailed monitoring levels ... 5053
View the automatic dashboard ... 5054
Costs ... 5054
Amazon OpenSearch Service insights ... 5054
Requirements ... 5055
Enable the managed collector ... 5055
Validate metrics collection ... 5055
Build a custom dashboard ... 5055
Costs ... 5055
Monitor across accounts and Regions ... 5056
CloudWatch cross-account observability ... 5061
Link monitoring accounts with source accounts ... 5064
Manage monitoring accounts and source accounts ... 5076
Cross-account cross-Region CloudWatch console ... 5079
Enabling cross-account cross-Region functionality ... 5080
(Optional) Integrate with AWS Organizations ... 5087

Troubleshooting 5087
Monitoring account permissions for cross-account access 5088
Disabling and cleaning up after using cross-account 5089
Cross-account cross-Region metrics centralization 5090
Metrics centralization concepts 5090
Centralized metrics 5091
Setting up metrics centralization 5092
Features supported with centralized metrics 5095
Monitoring and troubleshooting centralization rules 5096
Pricing 5097
Related resources 5097
Analyzing, optimizing, and reducing CloudWatch costs 5098
Analyze CloudWatch cost and usage data with Cost Explorer 5098
To visualize and analyze CloudWatch cost and usage data 5098
Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena 5104
To analyze cost and usage data with AWS Cost and Usage Reports and Athena 5105
Optimizing and reducing costs of CloudWatch metrics 5109
Custom metrics 5109
Detailed monitoring 5110
Embedded metrics 5111
API requests 5113
CloudWatch metric streams 5117
Optimizing and reducing costs of CloudWatch alarms 5118
Metric alarms 5118
Metrics Insights query alarms 5119
Composite alarms 5119
Optimizing and reducing costs of CloudWatch Container Insights 5121
Optimizing and reducing costs of CloudWatch Database Insights 5122
Optimizing and reducing costs of CloudWatch Logs 5123
AWS usage metrics 5130
Visualizing your service quotas and setting alarms 5130
AWS API usage metrics 5132
CloudWatch usage metrics 5140
SDK and API reference 5142
FAQs on CloudWatch supported protocols 5142

What is the AWS JSON protocol, and how does it differ from existing CloudWatch API requests and responses? ... 5142
What is the Smithy RPC v2 CBOR protocol, and how does it differ from existing CloudWatch API requests and responses? ... 5143
How do I get started with the newer protocols for CloudWatch? ... 5143
What are the risks of enabling these new protocols for my CloudWatch workloads? ... 5144
What if I am already on the latest AWS SDK version, but my open sourced solution does not support JSON or CBOR? ... 5145
What regions are supported for AWS JSON and CBOR protocols used in CloudWatch APIs ... 5145
What latency improvements can I expect when upgrading to the specified AWS SDK versions for CloudWatch using the newer protocols? ... 5145
Will the AWS Query protocol be deprecated? ... 5145
Where can I find more information about AWS JSON and Smithy RPC v2 CBOR protocols? ... 5145
Working with AWS SDKs ... 5146
Security ... 5147
Data protection ... 5148
Encryption at rest for OTel metrics ... 5149
Identity and access management ... 5158
Compliance validation ... 5313
Resilience ... 5314
Infrastructure security ... 5314
AWS Security Hub ... 5315
Interface VPC endpoints ... 5315
Security considerations for Synthetics canaries ... 5321
Code examples ... 5326
Basics ... 5327
Hello CloudWatch ... 5329
Learn the basics ... 5333
Actions ... 5507
Scenarios ... 5742
Creating a monitoring dashboard with function name as a variable ... 5743
Get started with alarms ... 5753
Manage custom metrics and alarms ... 5756
Monitor DynamoDB performance ... 5764

Run CPU stress tests on virtual machine instances using fault injection 5765
Send OpenTelemetry metrics and alarm on them with PromQL 5780
Using property variables in monitoring dashboards to monitor multiple serverless functions 5823
Logging API and console operations with AWS CloudTrail 5833
CloudWatch information in CloudTrail 5835
Example: CloudWatch log file entries 5836
CloudWatch data events in CloudTrail 5839
Query generation information in CloudTrail 5847
Amazon Q Developer operational investigations events in CloudTrail 5848
Example: Amazon Q Developer operational investigations log file entries 5849
Network Flow Monitor in CloudTrail 5852
Example: Network Flow Monitor log file entry 5853
Network Flow Monitor data plane events in CloudTrail 5856
Internet Monitor in CloudTrail 5859
Example: Internet Monitor log file entries 5860
CloudWatch Synthetics information in CloudTrail 5862
Example: CloudWatch Synthetics log file entries 5863
CloudWatch RUM information in CloudTrail 5867
Example: CloudWatch RUM log file entries 5867
CloudWatch RUM data plane events in CloudTrail 5872
Network Synthetic Monitor information in CloudTrail 5874
Example: Network Synthetic Monitor log file entries 5875
CloudWatch Observability Access Manager information in CloudTrail 5876
Example: CloudWatch Observability Access Manager log file entries 5877
CloudWatch Observability Admin information in CloudTrail 5878
Example: CloudWatch Observability Admin log file entries 5878
CloudWatch Application Signals information in CloudTrail 5879
Example: CloudWatch Application Signals log file entries 5880
CloudWatch Application Insights information in CloudTrail 5881
Example: CloudWatch Application Insights log file entries 5882
Tagging your CloudWatch resources 5884
Supported resources in CloudWatch 5884
Managing tags 5885
Tag naming and usage conventions 5885
Service quotas 5886

CloudWatch 5887
CloudWatch investigations (Amazon AI Operations) 5899
CloudWatch Application Insights 5899
CloudWatch Application Signals 5900
Internet Monitor 5904
Network Flow Monitor 5905
Network Synthetic Monitor 5906
CloudWatch Observability Access Manager 5906
CloudWatch Observability Admin 5910
CloudWatch Omni 5917
CloudWatch RUM 5925
Managing your CloudWatch service quotas 5925
Document history 5928

What is Amazon CloudWatch?

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time, and offers many tools to give you system-wide observability of your application performance, operational health, and resource utilization.

Topics

- Monitor applications and AI agents with CloudWatch Omni
- Operational visibility with metrics, alarms, and dashboards
- Application performance monitoring (APM)
- Infrastructure monitoring
- Collect, store, and query logs
- Use the CloudWatch agent to gather metrics, logs, and traces from Amazon EC2 fleets
- Cross-account monitoring
- OpenTelemetry support
- Solutions catalog
- Network and internet monitoring
- Billing and costs
- Amazon CloudWatch resources

Monitor applications and AI agents with CloudWatch Omni

CloudWatch Omni is an AI-powered observability experience in CloudWatch for your applications and AI agents. Omni discovers your services and their dependencies from your telemetry. Ask a question in plain language, and Omni writes the query for you, in SQL or PromQL (Prometheus Query Language). For AI agents, Omni scores response quality with evaluators, which catches failures that error rates do not show: an agent can return a well-formed response that is wrong. The scores appear alongside latency, errors, and token usage on the same trace.

Omni is built on CloudWatch and OpenTelemetry, and it does not replace anything you already run. Telemetry you already send to CloudWatch appears in Omni with nothing to reconfigure, and your existing metrics, alarms, and dashboards keep working unchanged. You reach Omni through a dedicated URL for your organization and sign in with the identities you already manage, or work in

the IDE extension for VS Code, Kiro, and Cursor while you develop AI agents. For more information, see *CloudWatch Omni*.

Operational visibility with metrics, alarms, and dashboards

Metrics collect and track key performance data at user-defined intervals. CloudWatch provides two types of metrics: OpenTelemetry Metrics (recommended) and CloudWatch Metrics (Classic). Many AWS services automatically report metrics into CloudWatch, and you can also publish your own custom metrics, either with OpenTelemetry (OTLP) or with the PutMetricData API.

Dashboards offer a unified view of your resources and applications with visualizations of your metrics and logs in a single location. You can also share dashboards across accounts and Regions for enhanced operational awareness. CloudWatch provides curated automatic dashboards for many AWS services, so that you don't have to build them yourself.

You can set up alarms that continuously monitor CloudWatch metrics against user-defined thresholds. They can automatically alert you to breaches of the thresholds, and can also automatically respond to changes in your resources' behavior by triggering automated actions.

Application performance monitoring (APM)

With Application Signals you can automatically detect and monitor your applications' key performance indicators like latency, error rates, and request rates without manual instrumentation or code changes. Application Signals also provides curated dashboards so you can begin monitoring with a minimum of setup.

CloudWatch Synthetics complements this by enabling you to proactively monitor your endpoints and APIs through configurable scripts called *canaries* that simulate user behavior and alert you to availability issues or performance degradation before they impact real users. You can also use CloudWatch RUM to gather performance data from real user sessions.

Use Service Level Objectives (SLOs) in CloudWatch to define, track, and alert on specific reliability targets for your applications, helping you maintain service quality commitments by setting error budgets and monitoring SLO compliance over time.

Infrastructure monitoring

Many AWS services automatically send basic metrics to CloudWatch for free. Services that send metrics are listed here. Additionally, CloudWatch provides additional monitoring capabilities for several key pieces of AWS infrastructure:

- Database Insights allows you to monitor database performance metrics in real time, analyze SQL query performance, and troubleshoot database load issues for AWS database services.
- Lambda Insights provides system-level metrics for Lambda functions, including memory and CPU utilization tracking, and cold start detection and analysis.
- Container Insights allows you to collect and analyze metrics from containerized applications, on Amazon ECS clusters, Amazon EKS clusters, and self-managed Kubernetes clusters on Amazon EC2.

Collect, store, and query logs

CloudWatch Logs offers a suite of powerful features for comprehensive log management and analysis. Logs ingested from AWS services and custom applications are stored in log groups and streams for easy organization. Use CloudWatch Logs Insights to perform interactive, fast queries on your log data, with a choice of three query languages including SQL and PPL. Use log anomaly detection to find unusual patterns in log events in a log group, which can indicate issues. Create metric filters to extract numerical values from logs and generate CloudWatch metrics, which you can use for alerting and dashboards. Set up subscription filters to process and analyze logs in real-time or route them to other services like Amazon S3 or Firehose.

Use the CloudWatch agent to gather metrics, logs, and traces from Amazon EC2 fleets

Use the CloudWatch agent to collect detailed system metrics about processes, CPU, memory, disk usage, and network performance from your fleets of Amazon EC2 instances and on-premises servers. You can also collect and monitor custom metrics from your applications, aggregate logs from multiple sources, and configure alarms based on the collected data. You can also use the agent to gather GPU metrics. The agent supports both Windows and Linux operating systems and can integrate with Systems Manager for centralized configuration management.

Cross-account monitoring

CloudWatch cross-account observability lets you set up a central monitoring account to monitor and troubleshoot applications that span multiple accounts. From the central account, you can view metrics, logs, and traces from source accounts across your organization. This centralized approach enables you to create cross-account dashboards, set up alarms that watch metrics from multiple accounts, and perform root-cause analysis across account boundaries. With CloudWatch cross-account observability, you can link source accounts either individually or link them automatically through AWS Organizations.

OpenTelemetry support

Amazon CloudWatch provides native OTLP endpoints for ingesting metrics, logs, and traces using the OpenTelemetry standard. You can collect telemetry using any OpenTelemetry-compatible SDK or collector and send it directly to CloudWatch without proprietary agents or format conversion. CloudWatch also supports PromQL for querying OpenTelemetry metrics. This open-standards approach lets you use the same instrumentation to send data to CloudWatch and third-party destinations, and query application and AWS infrastructure telemetry together. For more information, see OpenTelemetry.

Solutions catalog

CloudWatch offers a catalog of readily available configurations to help you quickly implement monitoring for various AWS services and common workloads, such as Java Virtual Machines (JVM), NVIDIA GPU, Apache Kafka, Apache Tomcat, and NGINX. These solutions provide focused guidance, including instructions for installing and configuring the CloudWatch agent, deploying pre-defined custom dashboards, and setting up related alarms.

Network and internet monitoring

CloudWatch provides comprehensive network and internet monitoring capabilities through CloudWatch Network Monitoring.

Internet Monitor uses AWS global networking data to analyze internet performance and availability between your applications and end users. With an internet monitor, you can identify or get notifications for increased latency or regional disruptions that impact your customers. Internet monitors work by analyzing your VPC flow logs to provide automated insights about network

traffic patterns and performance. You can also get suggestions for how to optimize application performance for your clients.

Network Flow Monitor displays network performance information gathered by lightweight software agents that you install on your instances. Using a flow monitor, you can quickly visualize packet loss and latency of your network connections over a time frame that you specify. Each monitor also generates a network health indicator (NHI), which tells you whether there were AWS network issues for the network flows tracked by your monitor during the time period that you're evaluating.

When you connect by using Direct Connect, you can use synthetic monitors in Network Synthetic Monitor to proactively monitor network connectivity by running synthetic tests between a VPC and on-premises endpoints. When you create a synthetic monitor, you specify probes by providing a VPC subnet and on-premises IP addresses. AWS creates and manages the infrastructure in the background that is required to perform round-trip time and packet loss measurements with the probes. These tests detect issues with connectivity, DNS, and latency before they impact your applications, so that you can take action to improve your end users' experience.

Billing and costs

For complete information about CloudWatch pricing, see Amazon CloudWatch Pricing.

For information that can help you analyze your bill and possibly optimize and reduce costs, see Analyzing, optimizing, and reducing CloudWatch costs.

Amazon CloudWatch resources

The following related resources can help you as you work with this service.

Resource	Description
Amazon CloudWatch FAQs	The FAQ covers the top questions developers have asked about this product.
AWS Developer Center	A central starting point to find documentation, code examples, release notes, and other information to help you build innovative applications with AWS.

Resource	Description
AWS Management Console	The console allows you to perform most of the functions of Amazon CloudWatch and various other AWS offerings without programming.
Amazon CloudWatch Discussion Forums	Community-based forum for developers to discuss technical questions related to Amazon CloudWatch.
AWS Support	The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
Amazon CloudWatch product information	The primary web page for information about Amazon CloudWatch.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse, and so on.

Quickstart: Send metrics, logs, and traces to CloudWatch with OpenTelemetry

In about 20 minutes, you send three signals from one application to CloudWatch: a metric, a log record, and a trace. You see the metric in Query Studio, the log in CloudWatch Logs, and the trace in Transaction Search.

This is the recommended path for new telemetry. You use the OpenTelemetry SDK, then the OpenTelemetry Collector, then the CloudWatch OpenTelemetry Protocol (OTLP) endpoints, and finally Query Studio, CloudWatch Logs, and Transaction Search. This path works the same way on Amazon EKS, Amazon ECS, Amazon EC2, and on-premises servers. The instrumentation that you write here doesn't change when you deploy.

When you finish, you have the following:

- A counter metric and a histogram metric, a log record, and a trace with a span, all emitted by a sample application
- A Collector that signs and forwards all three signals to CloudWatch
- A PromQL query and an alarm for the metric, the log record in CloudWatch Logs, and the trace in Transaction Search

The resources that you create in this quickstart might incur charges to your AWS account. Ingesting metrics, logs, and traces can each incur charges. For more information, see OTel metrics pricing and storage.

Choosing the right path

Use this quickstart for any new custom metric. Use a different approach only if one of the following applies to you.

If you	Use instead
Run in an AWS Region where OTLP metrics ingestion isn't available. For more information	`PutMetricData` or the embedded metric format. For more information, see Publish custom metrics (PutMetricData / EMF).

If you	Use instead
about supported Regions, see Supported AWS Regions.	
Already have alarms and dashboards on an existing custom namespace that you must keep	Keep publishing with `PutMetricData`, and migrate later. For more information, see Migrate from Classic to OTel metrics.
Need a metric from existing log events without changing code	A metric filter on the log group. For more information, see Creating metrics from log events using filters in the *Amazon CloudWatch Logs User Guide*.
Want request rate, errors, and latency without writing code	Application Signals. Add custom metrics only for business events. For more information, see Application Signals.

Prerequisites

Before you begin this quickstart, make sure that you have the following:

- An AWS account and access to Query Studio in the CloudWatch console in a supported AWS Region. This quickstart uses `aa-example-1`. Replace it with your Region everywhere that it appears.
- AWS credentials that the Collector can find. On AWS compute, use an AWS Identity and Access Management (IAM) role. For example, use an instance profile on Amazon EC2, use IAM roles for service accounts or Amazon EKS Pod Identity on Amazon EKS, or use a task role on Amazon ECS. When you test locally, use environment variables. To get credentials for the AWS CLI, see Getting set up with the AWS CLI in the *AWS Command Line Interface User Guide*.
- CloudWatch Transaction Search enabled in your Region. You can send traces to the OTLP traces endpoint only after you enable Transaction Search. For more information, see Transaction Search.
- Docker, to run the Collector.
- Python 3.9 or later, for the sample application. The Java, Go, .NET, and Node.js SDKs work the same way.

Important

Traces that you send to the X-Ray OTLP endpoint require Transaction Search to be enabled. This is a one-time, account-wide setting. It switches all X-Ray span ingestion into CloudWatch Logs collection mode, billed under CloudWatch pricing, with 1 percent of spans indexed for free. Metrics and logs don't require this setting.
To enable Transaction Search, run the following command.

```
aws xray update-trace-segment-destination --destination CloudWatchLogs --region
 aa-example-1
```

To verify, run the following command. The destination is ready when `Status` is `ACTIVE` and `Destination` is `CloudWatchLogs`.

```
aws xray get-trace-segment-destination --region aa-example-1
```

IAM policy for the Collector's role — The Collector needs permissions to send metrics, logs, and traces. Attach the AWS managed policy `CloudWatchAgentServerPolicy` to the Collector's role. This is the simplest option, and it grants everything that the Collector needs for all three signals.

Note

Sending metrics over OTLP uses the `cloudwatch:PutMetricData` permission, even though you never call that API. CloudWatch metrics don't support resource-level permissions, so policies that allow this action set `Resource` to `"*"`. The `CloudWatchAgentServerPolicy` managed policy already includes this permission, along with the permissions for logs and traces.

People who query metrics or create alarms also need the `cloudwatch:GetMetricData` and `cloudwatch:ListMetrics` permissions. For more information, see IAM permissions for PromQL.

Workloads outside AWS can use a bearer token instead of SigV4 for metrics, logs, and traces. Each service needs its own API key. For more information, including the Regions where traces support bearer tokens, see Setting up bearer token authentication for Metrics and Setting up bearer token authentication for traces.

Step 1: Instrumenting your application

Your application sends telemetry to a local Collector, not directly to CloudWatch. The SDK's OTLP exporter doesn't sign requests with SigV4, so CloudWatch rejects direct calls to the endpoints. The Collector handles signing, batching, and retries.

The following command installs the SDK and the OTLP HTTP exporter. The HTTP exporter package provides the metric, span, and log exporters that this quickstart uses.

```
pip install opentelemetry-sdk opentelemetry-exporter-otlp-proto-http
```

Create `app.py`. It records one counter (orders placed) and one histogram (checkout latency), wraps each checkout in a trace span, and writes a log record for each order. All three signals go to the local Collector.

```
import logging, random, time
from opentelemetry import metrics, trace
from opentelemetry._logs import set_logger_provider
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.sdk._logs import LoggerProvider, LoggingHandler
from opentelemetry.sdk._logs.export import BatchLogRecordProcessor
from opentelemetry.exporter.otlp.proto.http.metric_exporter import OTLPMetricExporter
from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
from opentelemetry.exporter.otlp.proto.http._log_exporter import OTLPLogExporter

# Static metadata goes on the resource, not on every data point
resource = Resource.create({
    "service.name": "checkout",
    "service.version": "1.4.0",
    "deployment.environment": "dev",
})

# Metrics: local Collector; exports every 60 seconds
metric_exporter = OTLPMetricExporter(endpoint="http://localhost:4318/v1/metrics")
reader = PeriodicExportingMetricReader(metric_exporter, export_interval_millis=60000)
metrics.set_meter_provider(MeterProvider(resource=resource, metric_readers=[reader]))
```

```
# Traces: local Collector
trace.set_tracer_provider(TracerProvider(resource=resource))
trace.get_tracer_provider().add_span_processor(
    BatchSpanProcessor(OTLPSpanExporter(endpoint="http://localhost:4318/v1/traces"))
)

# Logs: local Collector, bridged from the Python logging module
logger_provider = LoggerProvider(resource=resource)
logger_provider.add_log_record_processor(
    BatchLogRecordProcessor(OTLPLogExporter(endpoint="http://localhost:4318/v1/logs"))
)
set_logger_provider(logger_provider)
logging.getLogger().addHandler(LoggingHandler(logger_provider=logger_provider))
logging.getLogger().setLevel(logging.INFO)

meter = metrics.get_meter("checkout")
orders = meter.create_counter("orders_placed", unit="1", description="Orders placed")
latency = meter.create_histogram("checkout_duration", unit="s", description="Checkout
 latency")
tracer = trace.get_tracer("checkout")
log = logging.getLogger("checkout")

while True:
    try:
        payment = random.choice(["card", "wallet"])
        with tracer.start_as_current_span("checkout") as span:
            span.set_attribute("payment_method", payment)
            orders.add(1, {"payment_method": payment})
            duration = random.uniform(0.05, 1.2)
            latency.record(duration, {"payment_method": payment})
            log.info("Order placed using %s in %.3fs", payment, duration)
            time.sleep(1)
    except Exception as error:
        print(f"Error recording telemetry: {error}")
```

Important

Use attributes whose values come from a small, fixed set, such as `payment_method`. Never use order IDs, user IDs, or URLs that contain IDs as attribute values.

Step 2: Running the Collector

In this quickstart, you run the Collector locally and give it credentials through environment variables. In production, the Collector runs on your compute platform and uses an IAM role instead, as shown in Next steps.

> **Important**
>
> Sending metrics, logs, and traces to CloudWatch incurs charges. For more information, see OTel metrics pricing and storage.

The Collector receives all three signals from your application and exports each one to its CloudWatch OTLP endpoint. Use a distribution that includes the `sigv4auth` extension, such as the OpenTelemetry Collector Contrib distribution or the AWS Distro for OpenTelemetry (ADOT) Collector.

> **Note**
>
> The CloudWatch agent is an alternative. It can receive OpenTelemetry Protocol (OTLP) data and send it to CloudWatch, so you can use it instead of running a separate OpenTelemetry Collector. For more information, see Collect metrics and traces with OpenTelemetry.

The logs exporter sends to the log group and log stream that you set in the `x-aws-log-group` and `x-aws-log-stream` headers. This example uses a `/otel/quickstart` log group and a `default` stream. Create them in CloudWatch Logs first, or change the values to a log group and stream that already exist.

Create `collector.yaml`. In the following example, replace `aa-example-1` with your Region.

```
extensions:
  sigv4auth/metrics:
    service: monitoring        # always "monitoring" for metrics
    region: aa-example-1
  sigv4auth/logs:
    service: logs              # always "logs" for logs
    region: aa-example-1
  sigv4auth/traces:
    service: xray              # always "xray" for traces
```

```
    region: aa-example-1

receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

processors:
  memory_limiter:
    check_interval: 1s
    limit_percentage: 80         # Valid range: 1-100
    spike_limit_percentage: 20   # Valid range: 1-100
  batch:
    send_batch_size: 200         # Number of records to buffer before sending
    send_batch_max_size: 1000    # Maximum batch size; CloudWatch accepts at most 1,000
 metric data points for each request
    timeout: 10s                 # Maximum time to wait before sending a batch

exporters:
  otlphttp/metrics:            # the endpoints support HTTP only, not gRPC
    metrics_endpoint: https://monitoring.aa-example-1.amazonaws.com/v1/metrics
    compression: gzip
    auth:
      authenticator: sigv4auth/metrics
  otlphttp/logs:               # the endpoints support HTTP only, not gRPC
    logs_endpoint: https://logs.aa-example-1.amazonaws.com/v1/logs
    compression: gzip
    headers:
      x-aws-log-group: /otel/quickstart   # create this log group, or use an existing
 one
      x-aws-log-stream: default           # create this stream, or use an existing one
    auth:
      authenticator: sigv4auth/logs
  otlphttp/traces:             # the endpoints support HTTP only, not gRPC
    traces_endpoint: https://xray.aa-example-1.amazonaws.com/v1/traces
    compression: gzip
    auth:
      authenticator: sigv4auth/traces

service:
  extensions: [sigv4auth/metrics, sigv4auth/logs, sigv4auth/traces]
```

```
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/metrics]
    logs:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/logs]
    traces:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/traces]
```

The logs endpoint writes to an existing log group and log stream. Before you start the Collector, create the log group and stream named in the `x-aws-log-group` and `x-aws-log-stream` headers of your `collector.yaml` logs exporter. Otherwise, CloudWatch rejects your logs with `The specified log group does not exist`.

```
aws logs create-log-group --log-group-name /otel/quickstart --region aa-example-1
aws logs create-log-stream --log-group-name /otel/quickstart --log-stream-name default
 --region aa-example-1
```

Run the Collector locally with your credentials, and then start the application:

```
docker run --rm -p 4317:4317 -p 4318:4318 \
  -v "$PWD/collector.yaml:/etc/otelcol-contrib/config.yaml" \
  -e AWS_ACCESS_KEY_ID -e AWS_SECRET_ACCESS_KEY -e AWS_SESSION_TOKEN \
  otel/opentelemetry-collector-contrib:latest

python app.py
```

In production, pin a Collector version instead of using `latest`. On AWS compute, omit the `-e` flags. The Collector uses the role's credentials.

Step 3: Checking that your telemetry arrived

Data usually appears 1–2 minutes after the first export. Verify each signal in turn.

Metrics — In the CloudWatch console, open Query Studio and run each of the following queries. For more information, see Running a PromQL query in Query Studio.

To	PromQL
Confirm that the series exists	`orders_placed`
Show orders per second, by payment method	`sum by (payment_method) (rate(orders_placed[5m]))`
Show 95th percentile checkout latency	`histogram_quantile(0.95, sum(rate(checkout_duration[5m])))`

The raw `orders_placed` query returns one series for each payment method. Because a counter only increases, each series shows a cumulative total, as in the following example.

```
orders_placed{payment_method="card"}    512
orders_placed{payment_method="wallet"}  488
```

The `sum by (payment_method) (rate(orders_placed[5m]))` query converts those totals to a per-second rate. With the sample application, each series is near 0.5 per second, as in the following example.

```
{payment_method="card"}    0.51
{payment_method="wallet"}  0.49
```

Counters need `rate()` or `increase()`, because a raw counter only increases. You can query gauges directly. CloudWatch stores histograms as native histograms, so you query the metric name directly, without `_bucket` series or an `le` label. For more information, see Querying histogram metrics.

Logs — In the CloudWatch console, open CloudWatch Logs and open the `/otel/quickstart` log group (or the log group that you set in the `x-aws-log-group` header). Each order writes one log record. To search the records, use CloudWatch Logs Insights. For more information, see Analyze log data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*.

In Logs Insights, select the `/otel/quickstart` log group and run the following query to find your records:

```
fields @timestamp, severityText, body, traceId, spanId
```

```
| filter @message like /checkout/
| sort @timestamp desc
| limit 50
```

CloudWatch stores the OpenTelemetry log record as JSON, with resource attributes nested under `resource.attributes` (for example, `resource.attributes.service.name`), and the span's `payment_method` attribute is not a field on the log record — it appears in the span and in the log body. If you need to filter by a nested field, run a bare `fields @timestamp, @message | limit 50` query first to see the exact field paths.

Traces — In the CloudWatch console, open Transaction Search and open the Traces view. Each checkout produces one trace with a `checkout` span. If you don't see traces, confirm that Transaction Search is enabled in your Region.

Transaction Search stores spans in the `aws/spans` log group. You can search them in the CloudWatch Transaction Search console, filtered by the `checkout` service, or with a Logs Insights query on `aws/spans`. In Logs Insights, select the `aws/spans` log group and run the following query. This query filters on the top-level span name to avoid nested-field-path issues.

```
fields @timestamp, name, durationNano, attributes.payment_method, status.code, traceId
| filter name = "checkout"
| sort @timestamp desc
| limit 20
```

Each span carries the trace and span IDs that also appear on your log records, so you can pivot between a log line and its trace. CloudWatch enriches spans with Application Signals attributes such as `aws.local.service`.

If you don't see data after 5 minutes, see Troubleshooting telemetry export.

Step 4: Creating an alarm and a dashboard widget

Turn the latency query into an alarm and a dashboard widget, both from Query Studio.

To create an alarm and a dashboard widget

1. Open the CloudWatch console, and then open Query Studio.
2. In Query Studio, run the 95th percentile latency query from Step 3: Checking that your telemetry arrived.

3. From the actions menu, choose **Create alarm**. For more information, see Creating alarms from Query Studio.

   > **Note**
   >
   > If you receive an `AccessDenied` error, verify that the `CloudWatchAgentServerPolicy` managed policy is attached, or that your IAM policy includes the `cloudwatch:GetMetricData` and `cloudwatch:ListMetrics` permissions. For the required permissions, see Prerequisites.

4. Set the condition to greater than **`0.8`** seconds for 3 out of 3 periods.
5. Choose an Amazon SNS topic for notifications.
6. Return to Query Studio, choose **Add to dashboard**, and choose or create a dashboard for the checkout service. For more information, see Adding visualizations to dashboards.
7. (Optional) Add `sum(rate(orders_placed[5m]))` to the same dashboard.
8. (Optional) Turn on anomaly detection for the orders metric, so that a drop in orders alerts you without a fixed threshold. For more information, see Anomaly detection using PromQL.

Troubleshooting telemetry export

Check the Collector logs first. Export errors include the HTTP status code from CloudWatch.

Symptom	Likely cause	Fix
403 on metrics in the Collector logs	The role lacks `cloudwatch:PutMetricData`, or the `sigv4auth` service isn't `monitoring`.	Attach `CloudWatchAgentServerPolicy`, or fix the `service` value.
403 on logs in the Collector logs	The role lacks the logs permissions, or the `sigv4auth` service isn't `logs`.	Attach `CloudWatchAgentServerPolicy`, and set the `logs` `service` to `logs`.
4xx on logs, missing header	The logs exporter doesn't set the `x-aws-log-group` or `x-aws-log-stream` header.	Set both headers on the `otlphttp/logs` exporter to the target log group and stream.

Symptom	Likely cause	Fix
A log event is truncated or rejected	A single request to the logs endpoint is larger than 1 MB.	Reduce the batch size or the size of each log record so that each request stays under 1 MB.
Traces rejected	Transaction Search isn't enabled in the Region.	Enable CloudWatch Transaction Search, and then resend the traces.
403 on traces in the Collector logs	A bearer token was used for traces with a metrics or logs API key, or the X-Ray API key is invalid, inactive, or expired, or its IAM user isn't allowed `xray:CallWithBearerToken` or `xray:PutTraceSegments` .	Use an X-Ray API key with the required permissions, or use SigV4 (the `sigv4auth` extension with the `xray` service). For more information, see Setting up bearer token authentication for traces.
Connection errors to CloudWatch	The exporter is gRPC (`otlp`) instead of `otlphttp`, or the Region in the URL is wrong.	Use `otlphttp`, and use the same Region in the URL and in `sigv4auth` .
The application logs `connection refused`	The Collector isn't running, or port 4318 isn't published.	Start the Collector, and check `-p 4318:4318` .
400, too many data points	A batch has more than 1,000 data points.	Set `send_batch_max_size: 1000`.
400 on some series	A data point has more than 150 labels, or a label value is longer than 1,024 characters.	Drop or shorten attributes in a `transform` processor.
429	More than 500 requests per second for the account, or too many new series.	Send larger batches, and reduce the number of attribute values.

Symptom	Likely cause	Fix
Accepted, but not visible	The data point timestamp is more than 14 days in the past or more than 2 hours in the future.	Check the host clock.

For all limits, see Endpoint limits and restrictions.

Clean up resources

To avoid ongoing charges, delete the resources that you created in this quickstart when you no longer need them:

- Delete the alarm that you created.
- Remove the dashboard widget that you added.
- Delete the `/otel/quickstart` log group and its `default` log stream (or the log group and stream that you used) in CloudWatch Logs.
- Stop the Collector container and the sample application.
- Detach `CloudWatchAgentServerPolicy` from the Collector's role if you attached it only for this quickstart.

Keeping costs predictable

CloudWatch bills OTLP metrics per GB ingested, with 15 months of storage included. Logs and traces are also billed by the volume that you ingest, and Transaction Search has its own charges. Volume grows with the number of data points per series, the number of series, the number of log records, and the number of traces. For more information, see OTel metrics pricing and storage.

To keep costs predictable, keep the following practices in mind:

- **Export every 60 seconds.** Halving the interval roughly doubles the volume.
- **Count your series before you ship.** The number of series for a metric is the product of the number of distinct values of each attribute. For example, 2 payment methods, 3 Regions, and 5 status codes produce 30 series. Adding a user ID makes the number unbounded.

- **Put static metadata on the resource.** As shown in Step 1: Instrumenting your application, service name, version, and environment belong on the resource, not on each data point.
- **Use histograms where you need distributions.** A histogram data point carries more data than a counter or gauge data point. Use histograms for latency and sizes, not for everything.
- **Drop what you don't query.** With a `filter` processor in the Collector, you can remove unused metrics, log records, spans, or attributes before you're billed for them.

Next steps

Your instrumentation stays the same in production. Only where the Collector runs changes.

The following table shows where the Collector runs and which credentials to use in each environment.

Environment	Where the Collector runs	Credentials
Amazon EKS	A DaemonSet or Deployment. Applications send to its Service.	IAM roles for service accounts or Amazon EKS Pod Identity
Amazon ECS	A sidecar container in the task	Task role
Amazon EC2	A service on each instance	Instance profile
On-premises	A service on each host, or a central gateway	IAM Roles Anywhere, or a bearer token for metrics, logs, and traces

Before you go to production, also plan for the following:

- Configure the sending queue and retry settings on each exporter.
- Set up Collector health checks.
- Set up cross-account observability if telemetry from many accounts goes to one monitoring account. For more information, see CloudWatch cross-account observability.

For more ways to send telemetry, see Collect and send telemetry.

Collect and send telemetry

CloudWatch provides multiple methods to collect and send telemetry data from your applications and infrastructure. For new work, use OpenTelemetry. You send OpenTelemetry metrics, logs, and traces to the CloudWatch OTLP endpoints, either from the CloudWatch agent or from an OpenTelemetry collector. For more information, see OpenTelemetry.

You can also use the CloudWatch agent to collect metrics, logs, and traces from Amazon EC2 instances and on-premises servers. Separately, embedded metric format generates metrics from structured log data, CloudWatch Pipelines transform and route telemetry at scale, and managed Prometheus collectors discover and scrape Prometheus-compatible metrics from your AWS resources without running an agent.

Topics

- OpenTelemetry
- Collect metrics, logs, and traces using the CloudWatch agent
- Embedding metrics within logs
- CloudWatch pipelines
- Amazon CloudWatch managed Prometheus collectors

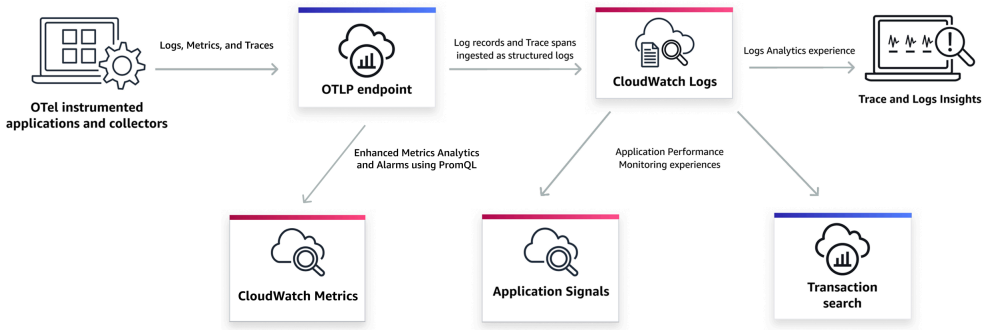
OpenTelemetry

> **Tip**
>
> To learn more about OpenTelemetry on CloudWatch, check out the Cloud Operations Enablement workshop and event series.

OpenTelemetry is an open-source observability framework that provides vendor-agnostic instrumentation for collecting metrics, logs, and traces from your applications. Amazon CloudWatch supports OpenTelemetry natively across all three signal types: metrics queryable with PromQL for flexible analytics, logs searchable with Logs Insights, and traces explorable with Transaction Search. Because all three signals share a common data model with consistent attributes, you can correlate across metrics, logs, and traces to move from detection to root cause faster.

You can send custom OTel metrics directly to CloudWatch and query them using PromQL alongside AWS vended metrics. For the resources and metrics that support this, see Supported AWS resources and metrics for OpenTelemetry enrichment. PromQL gives you flexible and scalable analytics, so you can build dashboards, set CloudWatch Alarms, and explore metrics in Query Studio, the native PromQL console experience. OpenTelemetry metrics also carry rich semantic labels and support higher granularity than traditional CloudWatch metrics. This granularity enables precise filtering, aggregation, and analysis across your applications and infrastructure. OTel logs that you send to CloudWatch are also available in LiveTail for real-time streaming.

For application performance monitoring, CloudWatch Application Signals provides a unified, application-centric view of your services and dependencies. Application Signals uses OTLP traces to help you monitor application health, triage issues, and identify the impact on end users. With Transaction Search, you can explore OTLP spans interactively, find transactions using attributes such as customer identifiers or order numbers, correlate transactions to business events such as failed payments, and trace interactions between application components to establish root cause.



Topics

- OTLP Endpoints
- Getting started
- Setting up bearer token authentication for Metrics
- Troubleshooting

OTLP Endpoints

OpenTelemetry Protocol (OTLP) is a general-purpose telemetry data delivery protocol designed for the OpenTelemetry. CloudWatch OpenTelemetry endpoints are HTTP 1.1 endpoints. You need to configure your OpenTelemetry collector to start sending open telemetry data to CloudWatch. For more information, see Getting started.

Logs endpoint

The logs endpoint follows the pattern `https://logs.AWS Region.amazonaws.com/v1/logs`. For example, for the `US West (Oregon) (us-west-2) Region`, the endpoint will be `https://logs.us-west-2.amazonaws.com/v1/logs`. You can use the above endpoint to forward logs to an existing `LogGroup` and `LogStream`. For more information on setting up `LogGroup` to ingest log data, see Amazon CloudWatch Logs concepts.

You must configure `LogGroup` and `LogStream` when you invoke CloudWatch Logs OpenTelemetry endpoint by setting `x-aws-log-group` and `x-aws-log-stream` HTTP headers to `LogGroup` and `LogStream` name respectively. For more information, see Getting started.

The endpoint supports the following authentication methods:

- **AWS Signature Version 4 (SigV4)** – Recommended for workloads running on AWS. Uses short-term credentials by using IAM roles. For more information, see AWS Signature Version 4 for API requests.
- **Bearer token (API key)** – For workloads running outside AWS or environments that do not support the AWS SDK. Requires bearer token authentication to be enabled on the target log group. For more information, see Setting up bearer token authentication for Logs.

When log event size exceed 1MB, CloudWatch Logs automatically truncates up to 10 fields, starting with the largest fields. Each field is truncated as needed to keep the total event size as close to 1MB as possible. The excess portions are stored as Large Log Objects (LLOs) and LLO reference system fields are added. Optionally, you can specify the field paths that need to be truncated by setting the `x-aws-truncatable-fields` HTTP header. The LLOs can be retrieved and streamed back using the `GetLogObject` API. For more information, see GetLogObject. Support of log events larger than 1MB and LLO experience is available in US East (N. Virginia) US West (Oregon), Europe (Frankfurt), Asia Pacific (Sydney), Asia Pacific (Mumbai), US East (Ohio), Europe (Ireland), Asia Pacific (Tokyo), and Asia Pacific (Singapore).

Metrics endpoint

The metrics endpoint follows the pattern `https://monitoring.AWS Region.amazonaws.com/v1/metrics`. For example, for the US West (Oregon) (us-west-2) Region, the endpoint will be `https://monitoring.us-west-2.amazonaws.com/v1/metrics`.

You need to configure your OpenTelemetry collector to start sending metrics to CloudWatch. The endpoint supports two authentication methods:

- **AWS Signature Version 4 (SigV4)**– The recommended authentication method using short-term credentials. For more information, see AWS Signature Version 4 for API requests.
- **Bearer token (API key)**– For workloads running outside AWS or environments that do not support the AWS SDK. Uses a long-lived API key passed in the `Authorization` header. For more information, see Setting up bearer token authentication for Metrics.

Traces endpoint

The traces endpoint follows the pattern `https://xray.AWS Region.amazonaws.com/v1/traces`. For example, for the US West (Oregon) (us-west-2) Region, the endpoint will be `https://xray.us-west-2.amazonaws.com/v1/traces`.

You need to configure your OpenTelemetry collector to start sending traces to CloudWatch. The endpoint supports two authentication methods:

- **AWS Signature Version 4 (SigV4)**– The recommended authentication method using short-term credentials. For more information, see AWS Signature Version 4 for API requests.
- **Bearer token (API key)**– For workloads running outside AWS or environments that do not support the AWS SDK. Uses a long-lived X-Ray API key passed in the `Authorization` header. For more information, including the supported Regions, see Setting up bearer token authentication for traces in the *X-Ray Developer Guide*.

RUM endpoint

The RUM endpoint follows the pattern `https://dataplane.rum.{AWS Region}.amazonaws.com/v1/rum`. For example, for the US West (Oregon) Region, the endpoint is `https://dataplane.rum.us-west-2.amazonaws.com/v1/rum`. This endpoint handles

client-side telemetry data (only traces and log records with `eventName`) for CloudWatch RUM applications.

To use this endpoint, you must create a RUM app monitor with Mobile platform (Android/ iOS) and use the code snippet generated to instrument your applications. The snippet will pull the RUM Mobile SDKs which are configured with this endpoint. You can configure the SDKs further for RUM to collect telemetry accordingly.

The endpoint supports both authenticated and unauthenticated requests. You can use AWS Signature Version 4 (SigV4) for authenticated requests, or resource-based policies to allow unauthenticated access from mobile applications.

To learn more about the authentication models as defined in their SDKs, see the following:

- iOS applications - AWS Distro for OpenTelemetry (ADOT) iOS SDK.
- Android applications - AWS Distro for OpenTelemetry (ADOT) Android SDK.

Endpoint limits and restrictions

The table lists the common endpoint limits and restrictions for metrics, traces, and logs.

Limit	Endpoint	Additional information
Required collector extension	sigv4authextension	To send metrics, logs, or traces to the OTLP endpoints, use sigv4authextension, or authenticate with a bearer token by using the bearertokenauth extension. See Setting up bearer token authentication for Metrics or Setting up bearer token authentication for Logs, or Setting

Limit	Endpoint	Additional information
		up bearer token authentication for traces.
Supported protocol	HTTP	The endpoint supports only HTTP and doesn't support gRPC
Supported OTLP versions	OTLP 1.x	
Payload format	binary, json	The endpoint accepts requests using binary and json formats
Compression methods	gzip, none	The endpoint only supports gzip and none compression methods

The table lists the endpoint limits and restrictions for traces.

Limit	Traces endpoint	Additional information
Maximum uncompressed bytes / request	5 MB	The OTLP endpoint will reject requests larger than 5MB when payload is uncompressed.
Maximum events / request	10,000 spans	The maximum number of spans in a batch is 10,000. Exceeding this limit

Limit	Traces endpoint	Additional information
		will cause rejection of the API call.
Single resource and scope size	16 KB	Each unique resource and corresponding scope should not exceed 16 KB of size. Exceeding this limit for any resource will cause rejection of the entire API call.
Single span maximum size	200 KB	Spans more than 200KB will be rejected by the endpoint.
Span created timestamps	2 hours in the future and 14 days in the past	None of the spans in the batch can be more than two hours in the future or more than 14 days in the past.
Maximum time gap in events / request	24 hours	

The table lists the endpoint limits and restrictions for metrics.

Limit	Metrics endpoint	Additional information	Error code
Maximum TPS	500	The maximum number of requests per second allowed per account.	429

Limit	Metrics endpoint	Additional information	Error code
Maximum new series creation rate	1,000,000	Maximum number of new series that can be created in a 10-minute window. This limit applies only to creating new metric names, not to ingesting data points for existing metrics.	429 - all metrics in the requests are throttled ; 200 - metrics in the requests are partially throttled
Maximum request size	5 MB	The maximum uncompressed size of the request.	413
Maximum datapoint count	1,000	The maximum number of datapoints sent in a single request. This number is a sum across ResourceMetrics, ScopeMetrics and Metrics.	400
Maximum metadata size	40 KB	The maximum combined size of all labels and label values accepted for a series per datapoint.	400 - all metrics in the requests are invalid; 200 - metrics in the requests are partially invalid

Limit	Metrics endpoint	Additional information	Error code
Maximum attribute value length	1024 characters	The maximum length of a string attribute value. Longer values are rejected with the following message: `Attribute string value exceeds maximum length: 1024`.	400
Maximum label count	150	Maximum number of labels across Resource/Scope/ Datapoint attributes per datapoint.	400 - all metrics in the requests are invalid; 200 - metrics in the requests are partially invalid
Metrics created timestamps	2 hours in the future and 14 days in the past	Metrics can be created with a timestamp that is at most 2 hours in the future and at most 14 days in the past.	400

The table lists the endpoint limits and restrictions for logs.

Limit	Logs endpoint	Additional information
Maximum uncompressed bytes / request	1 MB	The OTLP endpoint will reject requests larger than 1MB when payload is uncompressed.

Limit	Logs endpoint	Additional information
		The maximum request size is 1,048,576 bytes after decompression and deserialization of binary data serialized by Protocol buffers. This size is calculated as the sum of all event messages in UTF-8, plus 26 bytes for each log record.
	20 MB Available only in US East (N. Virginia) US West (Oregon), Europe (Frankfurt), Asia Pacific (Sydney), Asia Pacific (Mumbai), US East (Ohio), Europe (Ireland), Asia Pacific (Tokyo), and Asia Pacific (Singapore).	The maximum request size is 20MB (20,971,520 bytes) after the OTLP payload is decompressed and decoded from JSON format. For logs up to 1MB – These logs have full access to all CloudWatch Logs features including queries and live tail. For logs larger than 1MB – The excess portion is processed as Large Log Objects (LLOs).

Limit	Logs endpoint	Additional information
Request per second	5000	5000 transactions per second per account per Region You can request an increase to the per-second throttling quota by using the Service Quotas service.
Single resource and scope size	16 KB	Each unique resource and corresponding scope should not exceed 16 KB of size. Exceeding this limit for any resource will cause rejection of the entire API call.
Single LogEvent size	1 MB	LogEvent size is calculated as sum of sizes for each LogRecord, Scope and Resource. This quota can't be changed.

Limit	Logs endpoint	Additional information
Logs created timestamps	2 hours in future and 14 days old	The log records in the batch does not have to be in a chronological order. However, the log records in the batch cannot be more than 2 hours in the future and cannot be more than 14 days in the past. Also, none of the log records can be earlier than the retention period of the log group.
Maximum time gap in events / request	24 hours	
Maximum events / request	10,000 logs	The maximum number of log events in a batch is 10,000. Exceeding this limit will cause rejection of the API call.

Limit	Logs endpoint	Additional information
Maximum large log objects / request	1 log record Available in US East (N. Virginia) US West (Oregon), Europe (Frankfurt), Asia Pacific (Sydney), Asia Pacific (Mumbai), US East (Ohio), Europe (Ireland), Asia Pacific (Tokyo), and Asia Pacific (Singapore).	For content exceeding 1MB in a log event, the excess content is stored as LLOs. Limited to 1 log record per request.
Maximum large log objects / record	10 LLOs Available in US East (N. Virginia) US West (Oregon), Europe (Frankfurt), Asia Pacific (Sydney), Asia Pacific (Mumbai), US East (Ohio), Europe (Ireland), Asia Pacific (Tokyo), and Asia Pacific (Singapore).	A single log record can contain up to 10 LLOs.

Note

The account limits for Logs are shared across the SDK and the new Logs endpoint.

Getting started

To get started with OpenTelemetry in CloudWatch, you can choose from several options to send telemetry to the CloudWatch OTLP endpoints. For most customers, the CloudWatch agent is the recommended path. The CloudWatch agent is an AWS-managed OpenTelemetry Collector with CloudWatch components pre-built, and it provides the most integrated monitoring experience in CloudWatch.

You also have the flexibility to use an upstream OpenTelemetry Collector, build your own custom OpenTelemetry Collector, or send telemetry directly to the OTLP endpoint without a collector agent using the AWS Distro for OpenTelemetry (ADOT) SDK. Make an informed choice based on the feature support:

Feature	CloudWatch agent (recommended)	Upstream OpenTelemetry Collector	Custom OpenTelemetry Collector	Collectorless with AWS Distro for OpenTelemetry (ADOT) SDK
Search and analyze spans and trace summaries	Yes	Yes	Yes	Yes
Search and analyze logs summaries	Yes	Yes	Yes	No
CloudWatch entity correlation (resource linkage in the console and service map)	Yes	No	Yes	No
Runtime metrics correlated with your application. For example, JVM metrics	Yes	No	Yes	No
CloudWatch Enhanced Container Insights	Yes	No	No	No
CloudWatch Application Signals (Application	Yes	No	Yes	Yes

Feature	CloudWatch agent (recommended)	Upstream OpenTelemetry Collector	Custom OpenTelemetry Collector	Collectorless with AWS Distro for OpenTelemetry (ADOT) SDK
performance metrics, service discovery, and application map)				
Telemetry supported	Logs, Metrics, Traces	Logs, Metrics, Traces	Logs, Metrics, Traces	Metrics, Traces

> **Note**
>
> Make sure Transaction Search is enabled before you use the OTLP Endpoint for traces.

> **Note**
>
> AWS supports telemetry that you send to AWS destinations through the CloudWatch OTLP endpoints.

Topics

- Amazon CloudWatch agent
- OpenTelemetry Collector
- Build your own custom OpenTelemetry Collector
- Collect telemetry from compiled applications using eBPF
- Exporting collector-less telemetry using AWS Distro for OpenTelemetry (ADOT) SDK
- Managed Prometheus collectors

Amazon CloudWatch agent

The CloudWatch agent is built on the OpenTelemetry Collector, so you can use it to receive OpenTelemetry data and send it to the CloudWatch OTLP endpoints. In most cases, this is the recommended way to send OpenTelemetry data to CloudWatch, because a single agent can also power curated experiences such as CloudWatch Application Signals and CloudWatch Enhanced Container Insights. For more information about installing the agent, see Collect metrics, logs, and traces using the CloudWatch agent.

You can configure the agent to send OpenTelemetry data to the CloudWatch OTLP endpoints in two ways:

- **Using the agent configuration file (recommended)** – Add an `opentelemetry` section to your CloudWatch agent configuration file and enable the `otlp` source. The agent receives OTLP metrics, logs, and traces and forwards each signal to the correct CloudWatch OTLP endpoint. The agent sets the endpoints, the Region, and request signing for you, so you do not specify endpoint URLs or a `sigv4auth` extension. For the fields you can set, see CloudWatch agent configuration file: OpenTelemetry section.
- **Appending an OpenTelemetry collector configuration in YAML (advanced)** – Supply an OpenTelemetry collector configuration in YAML and append it to the agent's own configuration. Use this approach when you need components or pipeline topologies that the agent configuration file does not expose.

Send OpenTelemetry data using the agent configuration file

Add an `opentelemetry` section to your CloudWatch agent configuration file and include the `otlp` source under `collect`. When the agent starts with this configuration, it listens for OTLP data and forwards the received metrics, logs, and traces to the CloudWatch OTLP endpoints. For the fields you can set, their defaults, and the minimum agent version, see CloudWatch agent configuration file: OpenTelemetry section.

The following example configures the agent to receive OTLP data over gRPC and HTTP.

```
{
  "opentelemetry": {
    "collect": {
      "otlp": {
        "grpc_endpoint": "0.0.0.0:4317",
        "http_endpoint": "0.0.0.0:4318"
```

```
      }
    }
  }
}
```

Start the agent with this configuration the same way as any other agent configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -c
 file:/tmp/agent.json -s
```

> **Note**
>
> The agent signs requests to the CloudWatch OTLP endpoints with its own credentials. The `CloudWatchAgentServerPolicy` managed policy grants the permissions the agent needs to send metrics, logs, and traces to these endpoints.

Append an OpenTelemetry collector configuration in YAML

For pipelines that the agent configuration file does not expose, you can append an OpenTelemetry collector configuration in YAML. For the append procedure and the OpenTelemetry components that the agent supports, see Appending OpenTelemetry collector configuration files. The following examples show the YAML to append for each signal.

Configuration examples

The following examples send each signal to the corresponding CloudWatch OTLP endpoint using the `otlphttp` exporter and the `sigv4auth` extension. Each component and pipeline name uses a `/cwagent` suffix to avoid conflicts with pipelines that the agent creates automatically. Replace *`region`* with your AWS Region.

Metrics

```
receivers:
  otlp/cwagent:
    protocols:
      http:
        endpoint: 0.0.0.0:4318
processors:
```

```
  batch/cwagent: {}
exporters:
  otlphttp/cwagent:
    metrics_endpoint: https://monitoring.region.amazonaws.com/v1/metrics
    auth:
      authenticator: sigv4auth/cwagent
extensions:
  sigv4auth/cwagent:
    region: "region"
    service: "monitoring"
service:
  extensions: [sigv4auth/cwagent]
  pipelines:
    metrics/cwagent:
      receivers: [otlp/cwagent]
      processors: [batch/cwagent]
      exporters: [otlphttp/cwagent]
```

Logs

```
receivers:
  otlp/cwagent:
    protocols:
      http:
        endpoint: 0.0.0.0:4318
exporters:
  otlphttp/cwagent:
    logs_endpoint: https://logs.region.amazonaws.com/v1/logs
    headers:
      x-aws-log-group: my-log-group
      x-aws-log-stream: default
    auth:
      authenticator: sigv4auth/cwagent
extensions:
  sigv4auth/cwagent:
    region: "region"
    service: "logs"
service:
  extensions: [sigv4auth/cwagent]
  pipelines:
    logs/cwagent:
      receivers: [otlp/cwagent]
      exporters: [otlphttp/cwagent]
```

Traces

```
receivers:
  otlp/cwagent:
    protocols:
      http:
        endpoint: 0.0.0.0:4318
exporters:
  otlphttp/cwagent:
    traces_endpoint: https://xray.region.amazonaws.com/v1/traces
    auth:
      authenticator: sigv4auth/cwagent
extensions:
  sigv4auth/cwagent:
    region: "region"
    service: "xray"
service:
  extensions: [sigv4auth/cwagent]
  pipelines:
    traces/cwagent:
      receivers: [otlp/cwagent]
      exporters: [otlphttp/cwagent]
```

Example: send application metrics through the CloudWatch agent

This example sends application metrics through the CloudWatch agent to the CloudWatch OpenTelemetry Protocol (OTLP) endpoints. Instrument your application with an OpenTelemetry SDK and export OTLP to the agent that runs alongside it. The agent forwards the metrics to CloudWatch.

You might no longer need a separate collector

If you set up OpenTelemetry metrics collection before the agent's OTLP receiver was available, you might be running a standalone ADOT or OpenTelemetry collector alongside the agent that you no longer need. After you upgrade to agent version 1.300070.0 or later, you can remove that collector. The agent handles AWS SigV4 signing, Region, and endpoint selection, so you no longer need to configure them in a collector. Keep the ADOT SDK and auto-instrumentation that generate telemetry in your application. For more information about configuring the agent's OTLP receiver, see CloudWatch agent configuration file: OpenTelemetry section.

Prerequisites

Before you begin, complete the following prerequisites:

- Install CloudWatch agent version 1.300070.0 or later. Earlier versions do not support the agent's OTLP receiver. Some Linux package repositories offer an earlier version, so check the installed version before you continue. For more information, see CloudWatch agent configuration file: OpenTelemetry section.
- Attach the `CloudWatchAgentServerPolicy` managed policy, which grants the agent permission to send metrics, logs, and traces to the CloudWatch OTLP endpoints. The agent signs these requests with its own credentials.
- Instrument your application with an OpenTelemetry SDK or ADOT, and configure the application to export OTLP to the agent over gRPC (port 4317) or HTTP (port 4318).

Common steps

The following steps are the same regardless of where the agent runs.

1. Configure the agent to receive OTLP data. Add an `opentelemetry` section to your CloudWatch agent configuration file and enable the `otlp` source, as shown in Send OpenTelemetry data using the agent configuration file. The agent forwards the received metrics to the CloudWatch OTLP endpoints.
2. Point your application's OTLP exporter at the local agent instead of the public CloudWatch endpoint (for example, `http://localhost:4318`). The agent forwards the metrics for you. The following Python example uses the OpenTelemetry SDK to export metrics to the agent.

```
from opentelemetry import metrics
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader
from opentelemetry.exporter.otlp.proto.http.metric_exporter import OTLPMetricExporter

# Point at the local CloudWatch agent, which forwards to CloudWatch
exporter = OTLPMetricExporter(endpoint="http://localhost:4318/v1/metrics")
reader = PeriodicExportingMetricReader(exporter, export_interval_millis=60000)
provider = MeterProvider(metric_readers=[reader])
metrics.set_meter_provider(provider)

# Create and record a metric
meter = metrics.get_meter("my-app")
```

```
counter = meter.create_counter("http_requests_total", description="Total HTTP
 requests")
counter.add(1, {"method": "GET", "path": "/api/users", "status": "200"})
```

3. Verify that your metrics are arriving. Open the CloudWatch console, navigate to **Query Studio**, and run a PromQL query for your metric. For more information, see Verify metrics are arriving and Running PromQL queries in Query Studio.

Amazon EC2

Run the CloudWatch agent on the Amazon EC2 instance, and configure your application to export OTLP to the agent on the same instance. Set the application's OTLP exporter endpoint to `http://localhost:4317` for gRPC or `http://localhost:4318` for HTTP. The agent forwards the metrics to the CloudWatch OTLP endpoints.

Amazon ECS

On Amazon ECS, run the CloudWatch agent as a sidecar container in the same task as your application. You can also run the agent as a daemon service on each container instance. When the agent runs as a sidecar, your application exports OTLP to `localhost` within the task (port 4317 for gRPC or port 4318 for HTTP). When the agent runs as a daemon service, your application exports OTLP to the agent on the host. In both cases, the agent forwards the metrics to the CloudWatch OTLP endpoints.

Amazon EKS

On Amazon EKS, install or upgrade the CloudWatch Observability EKS add-on to version 6.6.0 or later. The add-on runs the CloudWatch agent in your cluster. Earlier versions do not expose the OTLP ports on the agent Service.

You must also supply the agent `opentelemetry` and `otlp` configuration to the add-on through its configuration values. The add-on does not enable the OTLP receiver by default. Supply the following configuration values to the add-on.

```
{
  "agent": {
    "config": {
      "opentelemetry": {
        "collect": {
          "otlp": {
            "grpc_endpoint": "0.0.0.0:4317",
```

```
            "http_endpoint": "0.0.0.0:4318"
          }
        }
      }
    }
  }
}
```

After the add-on rolls out, the `cloudwatch-agent` Service exposes ports 4317 and 4318. Your application pods export OTLP to the agent Service, and the agent forwards the metrics to the CloudWatch OTLP endpoints.

> **Supplying agent.config replaces the default configuration**
>
> When you supply `agent.config`, the add-on replaces its default agent configuration rather than merging with it. A configuration that contains only the `opentelemetry` section removes the Application Signals ports (4315, 4316, and 2000) and the `cwa-server` port (4311) from the Service. If you also use Application Signals or Container Insights, include those settings in the configuration that you supply so that you do not lose them.

OpenTelemetry Collector

The OpenTelemetry Collector is an open source, vendor-agnostic agent that receives, processes, and exports telemetry data. It acts as a central pipeline between your applications and Amazon CloudWatch, collecting metrics, logs, and traces from multiple sources and sending them to CloudWatch using the OpenTelemetry Protocol (OTLP).

Using the OpenTelemetry Collector with CloudWatch provides the following benefits:

- Collect telemetry from multiple applications and hosts through a single agent, reducing the number of connections to CloudWatch.
- Process and filter telemetry before sending it to CloudWatch, including adding or removing attributes, batching data, and sampling traces.
- Use the same collector configuration across AWS, on-premises, and other cloud environments, providing a consistent telemetry pipeline regardless of where your applications run.
- Send metrics to CloudWatch with rich labels that are available to query using the Prometheus Query Language (PromQL) in CloudWatch Query Studio.

Supported receivers

The OpenTelemetry Collector supports a wide range of receivers for ingesting telemetry data. You can use OpenTelemetry receivers such as the OTLP receiver for applications instrumented with OpenTelemetry SDKs, or Prometheus receivers to scrape metrics from existing Prometheus exporters. Common Prometheus receivers used with CloudWatch include:

- Prometheus receiver, for scraping any Prometheus-compatible endpoint
- Host Metrics receiver, for collecting system-level metrics from the host
- Kubernetes Cluster receiver, for collecting cluster-level metrics from the Kubernetes API server

You can configure multiple receivers in a single collector, allowing you to collect both OpenTelemetry and Prometheus metrics and send them to CloudWatch through the same pipeline. For the full list of available receivers, see the OpenTelemetry Collector repository.

Getting started

Prerequisite – If using the OTLP endpoint for tracing, make sure that Transaction Search is enabled.

Steps:

1. Download the latest release of the OpenTelemetry Collector distribution. For more information, see the OpenTelemetry Collector releases.
2. Install the OpenTelemetry Collector on your host. The collector runs on any operating system and platform. For more information, see Install the Collector.
3. Configure AWS credentials on your Amazon EC2 or on-premises host. The collector uses these credentials to authenticate with CloudWatch when sending telemetry data. See below for details.

   Setup IAM permissions for Amazon EC2

   Follow the below procedure to attach the `CloudWatchAgentServerPolicy` IAM policy to the IAM role of your Amazon EC2 instance.

   1. Open the IAM console at https://console.aws.amazon.com/iam/.
   2. Choose **Roles** and find and select the role used by your Amazon EC2 instance.
   3. Under the **Permissions** tab, choose **Add permissions**, **Attach policies**.
   4. Using the search box, search for `CloudWatchAgentServerPolicy` policy.
   5. Select the **CloudWatchAgentServerPolicy** policy and choose **Add permissions**.

Setup IAM permissions for on-premises hosts

You can create an IAM user that can be used to provide permissions to your on-premise hosts.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Users**, **Create User**.
3. Under **User details**,for **User name**, enter a name for the new IAM user. This is the sign-in name for AWS that will be used to authenticate your host.
4. Choose **Next**.
5. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
6. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user.
7. Choose **Next**.
8. On the **Review and create** page, make sure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is under the **Permissions summary**.
9. Choose **Create user**.
10. **Create and retrieve your AWS access key and secret key** – In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
11. On the user's page, choose the **Security credentials** tab.
12. Under the **Access keys** section, choose **Create access key**.
13. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
14. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.
15. For **Create access key Step 3,** select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.
16. Choose **Done**.
17. Configure your AWS credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Tip

For on-premises hosts that send metrics, logs, or traces, you can use bearer token authentication as a simpler alternative to configuring AWS access keys. Bearer tokens don't require the `aws configure` step or credential files on the host. For more information, see Setting up bearer token authentication for Metrics and Setting up bearer token authentication for traces.

4. Setup AWS credentials for your Amazon EKS or Kubernetes clusters. The easiest way to get started with Amazon EKS is to use the EKS OTel Container Insights add-on. If you prefer to use the OpenTelemetry Collector directly, follow the procedure below to set up AWS credentials for your Amazon EKS or Kubernetes clusters to send telemetry to CloudWatch.

   Setup IAM permissions for Amazon EKS

   1. Create an IAM OIDC identity provider for your cluster using the following command.

      ```
      eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} --region
       ${REGION} --approve
      ```

   2. Assign IAM roles to Kubernetes service account for OTel Collector using the following command.

      ```
      eksctl create iamserviceaccount \
      --name ${COLLECTOR_SERVICE_ACCOUNT}\
      --namespace ${NAMESPACE} \
      --cluster ${CLUSTER_NAME} \
      --region ${REGION} \
      --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
      --approve \
      ```

```
--override-existing-serviceaccounts
```

Setup IAM permissions for Kubernetes

1. Configure your AWS credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step. By default, the credential file is saved under */home/user/.aws/credentials.*.

   ```
   aws configure
   AWS Access Key ID [None]: ACCESS_KEY_ID
   AWS Secret Access Key [None]: SECRET_ACCESS_ID
   Default region name [None]: MY_REGION
   Default output format [None]: json
   ```

2. Edit OpenTelemetry Collector resource to add the newly created AWS credentials secret by using the command: `kubectl edit OpenTelemetryCollector` *otel_collector*

3. Using the file editor, add the AWS credentials into the OpenTelemetryCollector container by adding the following configuration to the top of the deployment. Replace the path */home/user/.aws/credentials* with the location of your local AWS credentials file.

   ```
    spec:
   volumeMounts:
   - mountPath: /rootfs
   volumeMounts:
   - name: aws-credentials
   mountPath: /root/.aws
   readOnly: true
   volumes:
   - hostPath:
   path: /home/user/.aws/credentials
   name: aws-credentials
   ```

5. Configure the OTLP exporter in your collector configuration to send telemetry to the CloudWatch endpoint. See examples below.

Authenticate with a bearer token (API key)

If you don't need to configure AWS credentials on the host – for example, when running on non-AWS infrastructure, other cloud providers, or CI/CD pipelines – you can use bearer token authentication instead of SigV4. Bearer tokens are supported for the metrics, logs, and traces endpoints (separate API keys are required for each service).

For setup instructions, see Setting up bearer token authentication for Metrics. For logs, see Setting up bearer token authentication for Logs. For traces, including the Regions where bearer tokens are supported, see Setting up bearer token authentication for traces in the *X-Ray Developer Guide*.

For a collector configuration example using bearer tokens, see the bearer token metrics example in the collector configuration examples below.

Collector configuration examples

Copy and paste the content below to configure your collector to send logs and traces to the OTLP endpoints.

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  otlphttp/logs:
    compression: gzip
    logs_endpoint: logs_otlp_endpoint
    headers:
      x-aws-log-group: ency_log_group
      x-aws-log-stream: default
    auth:
      authenticator: sigv4auth/logs

  otlphttp/traces:
    compression: gzip
    traces_endpoint: traces_otlp_endpoint
    auth:
      authenticator: sigv4auth/traces
```

```
extensions:
  sigv4auth/logs:
    region: "region"
    service: "logs"
  sigv4auth/traces:
    region: "region"
    service: "xray"

service:
  telemetry:
  extensions: [sigv4auth/logs, sigv4auth/traces]
  pipelines:
    logs:
      receivers: [otlp]
      exporters: [otlphttp/logs]
    traces:
      receivers: [otlp]
      exporters: [otlphttp/traces]
```

The following is an example to send logs and traces using sigv4 to us-east-1.

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  otlphttp/logs:
    compression: gzip
    logs_endpoint: https://logs.us-east-1.amazonaws.com/v1/logs
    headers:
      x-aws-log-group: MyApplicationLogs
      x-aws-log-stream: default
    auth:
      authenticator: sigv4auth/logs

  otlphttp/traces:
    compression: gzip
```

```
    traces_endpoint: https://xray.us-east-1.amazonaws.com/v1/traces
    auth:
      authenticator: sigv4auth/traces

extensions:
  sigv4auth/logs:
    region: "us-east-1"
    service: "logs"
  sigv4auth/traces:
    region: "us-east-1"
    service: "xray"

service:
  telemetry:
  extensions: [sigv4auth/logs, sigv4auth/traces]
  pipelines:
    logs:
     receivers: [otlp]
      exporters: [otlphttp/logs]
    traces:
      receivers: [otlp]
      exporters: [otlphttp/traces]
```

Note

Configure your OpenTelemetry SDKs to *always_on* sampling configuration to reliably record 100% spans and get full visibility into your critical applications with CloudWatch Application Signals. For more information, see an OpenTelemetry Java SDK sampler configuration example. For an example on setting up OpenTelemetry Collector with X-Ray OTLP endpoint, see the application signals demo repository.

Copy and paste the content below to configure your collector to send metrics to the OTLP endpoints.

```
receivers:
  otlp:
    protocols:
      http:
```

```
        endpoint: "0.0.0.0:4318"

processors:
  batch:
    send_batch_size: 200
    timeout: 10s

exporters:
  otlphttp:
    tls:
      insecure: false
    endpoint: metrics_otlp_endpoint
    auth:
      authenticator: sigv4auth

extensions:
  sigv4auth:
    service: "monitoring"
    region: "region"

service:
  extensions: [sigv4auth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]
```

The following is an example to send metrics using sigv4 to us-east-1.

```
receivers:
  otlp:
    protocols:
      http:
        endpoint: "0.0.0.0:4318"

processors:
  batch:
    send_batch_size: 200
    timeout: 10s

exporters:
```

```
  otlphttp:
    tls:
      insecure: false
    endpoint: "https://monitoring.us-east-1.amazonaws.com/v1/metrics:443"
    auth:
      authenticator: sigv4auth

extensions:
  sigv4auth:
    service: "monitoring"
    region: "us-east-1"

service:
  extensions: [sigv4auth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]
```

Metrics with bearer token authentication

The following is an example to send metrics using a bearer token to us-east-1. For setup instructions and security best practices, see Setting up bearer token authentication for Metrics.

```
extensions:
  bearertokenauth:
    filename: "/etc/otel/cw-api-key"

receivers:
  otlp:
    protocols:
      http:
        endpoint: "0.0.0.0:4318"

processors:
  batch:
    send_batch_size: 200
    timeout: 10s

exporters:
  otlphttp:
```

```
    tls:
      insecure: false
    endpoint: "https://monitoring.us-east-1.amazonaws.com/v1/metrics"
    auth:
      authenticator: bearertokenauth

service:
  extensions: [bearertokenauth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]
```

Important

Never hardcode API keys directly in collector configuration files. Use `filename` to read from a mounted secret, or `${env:VAR}` to read from an environment variable injected by your secrets manager.

Note

Unlike SigV4, bearer tokens don't require AWS credentials files, IAM roles, or IRSA configuration. The collector can run on any platform without AWS SDK dependencies.

Build your own custom OpenTelemetry Collector

You can build your own custom OpenTelemetry Collector to get the best application observability experience in CloudWatch with OpenTelemetry. In this setup, you need to build your own OpenTelemetry Collector with open source CloudWatch components.

Prerequisite

Make sure *Transaction Search* is enabled in CloudWatch. For more information, see Transaction Search.

Build your own collector

You can build your own collector with the following configuration to monitor your application in CloudWatch with OpenTelemetry. For more information, see Building a custom collector.

The common configuration for CloudWatch.

```
dist:
  name: otelcol-dev
  description: OTel Collector for sending telemetry to CloudWatch.
  output_path: ./otelcol-dev
extensions:
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/extension/
sigv4authextension v0.111.0
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/extension/awsproxy
 v0.113.0
exporters:
  - gomod: go.opentelemetry.io/collector/exporter/otlpexporter v0.111.0
  - gomod: go.opentelemetry.io/collector/exporter/otlphttpexporter v0.111.0
receivers:
  - gomod: go.opentelemetry.io/collector/receiver/otlpreceiver v0.111.0
```

Additional configuration for traces.

```
# Enable Tracing
dist:
  name: otelcol-dev
  description: OTel Collector for sending telemetry to CloudWatch.
  output_path: ./otelcol-dev
extensions:
    #Include common configurations and your custom extensions

exporters:
    #Include common configurations and your custom extensions

receivers:
  - gomod: go.opentelemetry.io/collector/receiver/otlpreceiver v0.111.0
processors:
  - gomod: github.com/amazon-contributing/opentelemetry-collector-contrib/processor/
awsapplicationsignalsprocessor v0.113.0
```

```
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/processor/
resourcedetectionprocessor v0.113.0
  - gomod: github.com/open-telemetry/opentelemetry-collector-contrib/processor/
metricstransformprocessor v0.113.0
replaces:
  - github.com/open-telemetry/opentelemetry-collector-contrib/internal/aws/awsutil
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/internal/
aws/awsutil v0.113.0
  - github.com/open-telemetry/opentelemetry-collector-contrib/internal/aws/cwlogs
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/internal/
aws/cwlogs v0.113.0
  - github.com/open-telemetry/opentelemetry-collector-contrib/exporter/awsemfexporter
 v0.113.0 => github.com/amazon-contributing/opentelemetry-collector-contrib/exporter/
awsemfexporter v0.113.0
  - github.com/openshift/api v3.9.0+incompatible => github.com/openshift/api
 v0.0.0-20180801171038-322a19404e37
```

Note

Note the following:

- After the collector is built, deploy and configure the custom collector in a host or kubernetes environment by following the procedure under OpenTelemetry Collector.
- For information on setting up custom OpenTelemetry collector with Application Signals Processor, see an Application Signals custom configurationexample. Application Signals Processor only supports the latest versions of the OpenTelemetry Collectors for custom builds. For information on the supported versions, see opentelemetry-collector-contrib repository.

Collect telemetry from compiled applications using eBPF

You cannot instrument some applications by adding an OpenTelemetry SDK to the source. Examples include compiled binaries (such as Go, Rust, or C/C++), third-party or vendor binaries whose source you do not own, and applications whose libraries an in-process agent cannot attach to. For these applications, you can use eBPF-based auto-instrumentation to generate OpenTelemetry telemetry without changing application code. You then send that telemetry to CloudWatch through the CloudWatch agent.

Important

The CloudWatch agent is the OTLP receiver and exporter (the collection layer). It does not instrument your application. You pair the agent with a separate eBPF instrumentation tool that produces OTLP data. The open source OpenTelemetry project for this is OpenTelemetry eBPF Instrumentation (OBI) on the OpenTelemetry website. eBPF attaches probes at the kernel, network, and TLS layers. As a result, it captures HTTP, HTTPS, and gRPC traffic and produces rate, errors, and duration (RED) metrics and trace spans without per-library code changes. For the languages, runtime versions, and libraries that OBI supports, see the OBI documentation. Coverage might vary by language and library, including which client and server spans it produces. Validate the specific frameworks your application uses before you rely on this approach.

Understanding how it works

The telemetry flows through the following pipeline.

1. eBPF auto-instrumentation generates telemetry without a code change and emits OTLP.
2. The CloudWatch agent, which is built on the OpenTelemetry Collector, receives the OTLP data. For more information, see Amazon CloudWatch agent.
3. The agent forwards traces to AWS X-Ray (Transaction Search) and metrics to CloudWatch.

Determining when to use this approach

Consider this approach in the following cases:

- You have compiled binaries that you cannot add an SDK to.
- You have vendor or third-party code that you do not own.
- You must keep an existing third-party monitoring agent in place.
- You want RED metrics and traces without a code change or a redeploy of the application.

Completing the prerequisites

Before you begin, complete the following prerequisites:

- Install CloudWatch agent version 1.300070.0 or later. Earlier versions do not support the agent's OTLP receiver. On Amazon EKS, use the CloudWatch Observability EKS add-on version 6.6.0 or later so that the agent Service exposes the OTLP ports.
- Attach the `CloudWatchAgentServerPolicy` managed policy. The agent signs requests to the CloudWatch OTLP endpoints with its own credentials.
- Enable Transaction Search so that spans reach the X-Ray OTLP endpoint. For more information, see Transaction Search.
- The eBPF instrumentation tool runs on Linux and requires a kernel with eBPF support and a set of Linux capabilities (such as CAP_BPF). The exact capabilities depend on the instrumentation tool and which features you enable, so grant the least privilege the tool documents rather than running it fully privileged. See the OBI documentation.
- If the agent runs in a subnet without internet egress, make sure it can reach every endpoint it exports to. These endpoints are CloudWatch for metrics and X-Ray for traces (which Transaction Search uses). If you use the logs path, the agent must also reach CloudWatch Logs. In a private network, you typically provide this reachability with interface VPC endpoints (AWS PrivateLink). For the CloudWatch interface VPC endpoints, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints. For the X-Ray VPC endpoint, see Using X-Ray with interface VPC endpoints.

Step 1: Configure the CloudWatch agent to receive OTLP data

Add an `opentelemetry` section to your CloudWatch agent configuration file and include the `otlp` source under `collect`. Bind the receiver to `127.0.0.1` so that the agent accepts OTLP only from the local host. In containerized deployments where producers run outside the agent's own container network—for example an Amazon ECS sidecar, or the Amazon EKS add-on's `cloudwatch-agent` Service—set the endpoints to `0.0.0.0` instead. The task, pod, or Service boundary then controls what can reach the ports.

```
{
  "opentelemetry": {
    "collect": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
```

```
}
```

The agent forwards traces to X-Ray and metrics to CloudWatch automatically. For the full agent OTLP configuration, see Amazon CloudWatch agent.

Run one OTLP receiver per host

Run a single OTLP receiver—the CloudWatch agent—for each host network namespace, and have every telemetry producer (your eBPF instrumentation and any OpenTelemetry SDKs) export to it. Only one process can bind the default OTLP ports (4317 and 4318) in a given network namespace, so a second collector that also uses the defaults fails to start. If you previously ran a separate OpenTelemetry or ADOT collector, remove it when you move collection to the agent. The eBPF instrumentation is a producer that exports OTLP, so it does not open the OTLP receiver ports. On Amazon EKS, each pod has its own network namespace, so this conflict occurs only when a pod uses host networking or a host port. If two receivers must coexist, the ports are configurable—set `grpc_endpoint` and `http_endpoint` to non-default values.

Step 2: Point the eBPF instrumentation at the agent

Configure the eBPF tool's OTLP exporter to send to the local agent, using gRPC on port 4317 or HTTP on port 4318 (for example, `http://localhost:4318`). The following deployment notes apply:

- **Amazon EC2**—Run the agent and the eBPF tool on the same instance, and export OTLP to `localhost`.
- **Amazon EKS**—The CloudWatch Observability EKS add-on runs the agent and exposes ports 4317 and 4318 on the `cloudwatch-agent` Service. Run the eBPF instrumentation as a DaemonSet and point its exporter at that Service.

For the eBPF tool's own configuration, see the OBI documentation.

Step 3: Verify your telemetry

Verify that your telemetry is arriving:

- **Metrics**—Open the CloudWatch console, go to **Query Studio**, and run a PromQL query for your metric. OpenTelemetry metric names keep their dots in CloudWatch Query Studio, so query by

the metric's OpenTelemetry name with the __name__ matcher rather than a Prometheus-style normalized name—for example, {__name__="http.server.request.duration"} (the actual name depends on what your instrumentation emits).

- **Traces and spans**—Use Transaction Search. For more information, see Transaction Search.

> **Running alongside an existing monitoring agent**
>
> eBPF-based instrumentation runs out of process and does not inject into the application process. As a result, it can typically run alongside an existing in-process monitoring agent that does not instrument the same compiled runtime. Because eBPF tools require elevated privileges and some monitoring agents also use eBPF, validate resource usage and probe behavior in a test environment before you deploy to production.

Exporting collector-less telemetry using AWS Distro for OpenTelemetry (ADOT) SDK

You can use the ADOT SDKs to go collector-less and to send metrics, traces and logs directly to the OTLP endpoints.

> **Note**
>
> Application Signals includes Transaction Search capabilities. To avoid duplicate costs, disable Application Signals in the ADOT SDK and keep `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` set to false (default setting).

Topics

- Prerequisite
- Set up IAM permissions for your role
- Configure your credentials providers
- Enabling ADOT SDKs

Prerequisite

If you are using traces, make sure Transaction Search is enabled to send spans to the X-Ray OTLP endpoint. For more information, see Getting started with Transaction Search.

Set up IAM permissions for your role

Follow these steps to attach the necessary IAM policies to your role:

Traces:

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** and find and select your configured role.
3. Under the **Permissions** tab, choose **Add permissions**, then **Attach policies**.
4. Using the search box, search for the `AWSXrayWriteOnlyPolicy`.
5. Select the `AWSXrayWriteOnlyPolicy` policy and choose **Add permissions**.

Logs:

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles** and find and select your configured role.
3. Under the **Permissions** tab, choose **Add permissions**, then **Create inline policy**.
4. Select **CloudWatch Logs** for the service and under **Actions allowed**, filter and select:

```
logs:PutLogEvents
logs:DescribeLogGroups
logs:DescribeLogStreams
```

5. The following is an example IAM policy that grants the required permissions:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchLogsAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"

            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:*"
            ]
        }
    ]
}
```

Configure your credentials providers

ADOT uses the AWS SDKs to automatically discover valid credentials required to export your telemetry data to AWS. For guidance on configuring credentials for your specific environment, refer to the documentation on how the AWS SDKS retrieves credentials.

Setting up IAM credentials for on-premise hosts:

Configure your AWS credentials in your on-premises host by entering the following command. Replace **ACCESS_KEY_ID** and **SECRET_ACCESS_KEY** with the credentials for your configured IAM role or user.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Enabling ADOT SDKs

You can enable logs and traces for your application to be sent directly to the OTLP endpoints from AWS Distro for OpenTelemetry (ADOT) SDK on Java, Node.js, Python, and .Net.

> **Note**
>
> When you send traces directly to the OTLP endpoint without using a collector, the SDK defaults to `parentbased_always_on`, which samples 100% of traces. This can result in up to 20x higher ingestion volume and costs compared to the recommended 5% sampling rate.

If you plan to use Transaction Search for 100% trace visibility, no action is needed. If you want to moderate trace ingestion, configure a local sampler by setting the following environment variables before starting your application:

```
export OTEL_TRACES_SAMPLER="parentbased_traceidratio"
export OTEL_TRACES_SAMPLER_ARG="0.05"
```

The example above samples 5% of traces. Adjust the value of `OTEL_TRACES_SAMPLER_ARG` between `0` and `1` to set the sampling rate appropriate for your workload.
For more information about sampler configuration options, see OTEL_TRACES_SAMPLER in the OpenTelemetry documentation.

Java

Note

You must use **ADOT Java Agent version 2.11.2 or later** for these features to be available.

1. Download the latest version of the AWS Distro for OpenTelemetry Java auto-instrumentation agent. You can download the latest version by using this command:

   ```
   curl -L -O https://github.com/aws-observability/aws-otel-java-instrumentation/
   releases/latest/download/aws-opentelemetry-agent.jar
   ```

   You can view information about all released versions at aws-otel-java-instrumentation Releases.
2. To enable the different exporter that directly sends telemetry to the OTLP endpoints and to optimize benefits, use the following environment variables before you start your application:

   Traces:

   - Set `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` to specify the X-Ray OTLP traces endpoint: `https://xray.[AWSRegion].amazonaws.com/v1/traces`

For example:

```
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-
west-2.amazonaws.com/v1/traces"
```

- Set `OTEL_TRACES_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_EXPORTER_OTLP_TRACES_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_RESOURCE_ATTRIBUTES` variable to specify the following information as key-value pairs. These environment variables are used by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.:
  - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.
  - (Optional) `deployment.environment` sets the environment that the application runs in. This will be displayed as the **Hosted In** environment of your application.
  - (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace `$YOUR_APPLICATION_LOG_GROUP` with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Logs:

- Set `OTEL_EXPORTER_OTLP_LOGS_ENDPOINT` to specify the CloudWatch OTLP logs endpoint: `https://logs.[`*`AWSRegion`*`].amazonaws.com/v1/logs`

  For example:

```
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT="https://logs.us-west-2.amazonaws.com/
v1/logs"
```

- Set `OTEL_EXPORTER_OTLP_LOGS_HEADERS` to specify the log group and log stream (note: these must created **before** running ADOT) you want to export your logs to: `x-aws-log-group=[CW-LOG-GROUP-NAME],x-aws-log-stream=[CW-LOG-STREAM-NAME]`

  For example:

```
export OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-
stream=default
```

- Set `OTEL_LOGS_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_EXPORTER_OTLP_LOGS_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).

3. Set `JAVA_TOOL_OPTIONS` to specify the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example:

```
export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
export AWS_ADOT_JAVA_INSTRUMENTATION_PATH="./aws-opentelemetry-agent.jar"
```

4. Your application should now be running with ADOT Java instrumentation and will generate spans and logs. Spans will be stored in the `aws/spans` CloudWatch log group, while logs will be stored in the log group specified in the `OTEL_EXPORTER_OTLP_LOGS_HEADERS` header. You can also view the logs and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
5. Start your application with the environment variables listed in the previous step. The following is an example of a starting script.

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_TRACES_EXPORTER=otlp \
OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces \
OTEL_LOGS_EXPORTER=otlp \
```

```
OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=https://logs.us-east-1.amazonaws.com/v1/logs \
OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-
stream=default
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
java -jar $MY_JAVA_APP.jar
```

Node.js

> **Note**
>
> You must use **ADOT JavaScript version 0.7.0 or later** for these features to be available.

1. Download the latest version of the AWS Distro for OpenTelemetry JavaScript auto-instrumentation. Install it by running the following command.

   ```
   npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
   ```

   You can view information about all released versions at aws-otel-js-instrumentation Releases.
2. To enable the different exporter that directly sends telemetry to the OTLP endpoints and to optimize benefits, use the following environment variables before you start your application:

   Traces:

   - Set `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` to specify the X-Ray OTLP traces endpoint: `https://xray.[AWSRegion].amazonaws.com/v1/traces`

     For example:

     ```
     export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-
     west-2.amazonaws.com/v1/traces"
     ```
   - Set `OTEL_TRACES_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).
   - Set `OTEL_EXPORTER_OTLP_TRACES_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).

- Set `OTEL_RESOURCE_ATTRIBUTES` variable to specify the following information as key-value pairs. These environment variables are used by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.:
  - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.
  - (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application.
  - (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace `$YOUR_APPLICATION_LOG_GROUP` with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Logs:

- Set `OTEL_EXPORTER_OTLP_LOGS_ENDPOINT` to specify the CloudWatch OTLP logs endpoint: `https://logs.[`*`AWSRegion`*`].amazonaws.com/v1/logs`

  For example:

  ```
  export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT="https://logs.us-west-2.amazonaws.com/v1/logs"
  ```

- Set `OTEL_EXPORTER_OTLP_LOGS_HEADERS` to specify the log group and log stream (note: these must created **before** running ADOT) you want to export your logs to: `x-aws-log-group=[CW-LOG-GROUP-NAME],x-aws-log-stream=[CW-LOG-STREAM-NAME]`

  For example:

```
export OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-
stream=default
```

- Set `OTEL_LOGS_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_EXPORTER_OTLP_LOGS_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).

3. Your application should now be running with ADOT JavaScript instrumentation and will generate spans and logs. Spans will be stored in the `aws/spans` CloudWatch log group, while logs will be stored in the log group specified in the `OTEL_EXPORTER_OTLP_LOGS_HEADERS` header. You can also view the logs and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
4. Start your application with the environment variables listed in the previous step. The following is an example of a starting script.

   Replace `$SVC_NAME` with your application name. This will be displayed as the name of the application.

```
OTEL_METRICS_EXPORTER=none \
OTEL_TRACES_EXPORTER=otlp \
OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces \
OTEL_LOGS_EXPORTER=otlp \
OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=https://logs.us-east-1.amazonaws.com/v1/logs \
OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-
stream=default
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
node --require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/register'
 your-application.js
```

Python

> **Note**
>
> You must use **ADOT Python version 0.10.0 or later** and have **`botocore`** installed for these features to be available.

1. Download the latest version of the AWS Distro for OpenTelemetry Python auto-instrumentation. Install it by running the following command.

   ```
   pip install aws-opentelemetry-distro
   ```

   You can view information about all released versions at aws-otel-python-instrumentation Releases.

2. To enable the different exporter that directly sends telemetry to the OTLP endpoints and to optimize benefits, use the following environment variables before you start your application:

   For both traces and logs configurations, you must set the following environment variables:

   - `OTEL_PYTHON_DISTRO` to `aws_distro`
   - `OTEL_PYTHON_CONFIGURATOR` to `aws_configurator`

   Traces:

   - Set `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` to specify the X-Ray OTLP traces endpoint: `https://xray.[`*`AWSRegion`*`].amazonaws.com/v1/traces`

     For example:

     ```
     export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://xray.us-
     west-2.amazonaws.com/v1/traces"
     ```

   - Set `OTEL_TRACES_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).

- Set `OTEL_EXPORTER_OTLP_TRACES_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_RESOURCE_ATTRIBUTES` variable to specify the following information as key-value pairs. These environment variables are used by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.:
  - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.
  - (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application.
  - (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace `$YOUR_APPLICATION_LOG_GROUP` with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Logs:

- Set `OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED` to `true`
- Set `OTEL_EXPORTER_OTLP_LOGS_ENDPOINT` to specify the CloudWatch OTLP logs endpoint: `https://logs.[AWSRegion].amazonaws.com/v1/logs`

  For example:

```
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT="https://logs.us-west-2.amazonaws.com/
v1/logs"
```

- Set `OTEL_EXPORTER_OTLP_LOGS_HEADERS` to specify the log group and log stream (note: these must created **before** running ADOT) you want to export your logs to: `x-`

`aws-log-group=[CW-LOG-GROUP-NAME],x-aws-log-stream=[CW-LOG-STREAM-NAME]`

For example:

```
export OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-stream=default
```

- Set `OTEL_LOGS_EXPORTER` to `otlp` (this is optional and is the default value if this environment variable is not set).
- Set `OTEL_EXPORTER_OTLP_LOGS_PROTOCOL` variable to `http/protobuf` (this is optional and is the default value if this environment variable is not set).

3. Your application should now be running with ADOT Python instrumentation and will generate spans and logs. Spans will be stored in the `aws/spans` CloudWatch log group, while logs will be stored in the log group specified in the `OTEL_EXPORTER_OTLP_LOGS_HEADERS` header. You can also view the logs and metrics correlated with your spans in the CloudWatch Traces and Metrics Console.
4. Start your application with the environment variables listed in the previous step. The following is an example of a starting script.

   Replace `$SVC_NAME` with your application name. This will be displayed as the name of the application.

   Replace `$PYTHON_APP` with the location and name of your application.

```
OTEL_METRICS_EXPORTER=none \
OTEL_TRACES_EXPORTER=otlp \
OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/traces \
OTEL_LOGS_EXPORTER=otlp \
OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=https://logs.us-east-1.amazonaws.com/v1/logs \
OTEL_EXPORTER_OTLP_LOGS_HEADERS=x-aws-log-group=MyLogGroup,x-aws-log-stream=default
OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
opentelemetry-instrument python $MY_PYTHON_APP.py
```

.Net

> **Note**
>
> - **ADOT .NET version 1.8.0 or later** is required for these features to be available.
> - Compression is currently not supported.
> - Logs is currently not supported.

Download the latest version of the AWS Distro for OpenTelemetry .NET auto-instrumentation package. You can view information about all released versions at aws-otel-dotnet-instrumentation Releases.

To enable the different exporters that directly sends telemetry to the OTLP endpoints and to optimize benefits, use the following environment variables before you start your application (replace `dotnet-service-name` in the `OTEL_RESOURCE_ATTRIBUTES` environment variable with the service name of your choice):

Traces:

- You **MUST** set `OTEL_TRACES_EXPORTER` to none
- You **MUST** set `OTEL_AWS_SIG_V4_ENABLED` to `true`

  This feature is **not enabled automatically in .NET**. The environment variable is required specifically to identify this use case within the .NET instrumentation. This requirement is unique to .NET and does **not apply to other supported languages**.

The following is an example setup for Linux.

```
export INSTALL_DIR=OpenTelemetryDistribution
export CORECLR_ENABLE_PROFILING=1
export CORECLR_PROFILER={918728DD-259F-4A6A-AC2B-B85E1B658318}
export CORECLR_PROFILER_PATH=${INSTALL_DIR}/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so
export DOTNET_ADDITIONAL_DEPS=${INSTALL_DIR}/AdditionalDeps
export DOTNET_SHARED_STORE=${INSTALL_DIR}/store
export DOTNET_STARTUP_HOOKS=${INSTALL_DIR}/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll
```

```
export OTEL_DOTNET_AUTO_HOME=${INSTALL_DIR}

export
 OTEL_DOTNET_AUTO_PLUGINS="AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
export OTEL_TRACES_EXPORTER=none
export OTEL_AWS_SIG_V4_ENABLED=true

export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
export OTEL_METRICS_EXPORTER=none
export OTEL_LOGS_EXPORTER=none
export OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces
```

The following is an example setup for Windows Server:

```
$env:INSTALL_DIR = "OpenTelemetryDistribution"
$env:CORECLR_ENABLE_PROFILING = 1
$env:CORECLR_PROFILER = "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
$env:CORECLR_PROFILER_PATH = Join-Path $env:INSTALL_DIR "win-x64/
OpenTelemetry.AutoInstrumentation.Native.dll"
$env:DOTNET_ADDITIONAL_DEPS = Join-Path $env:INSTALL_DIR "AdditionalDeps"
$env:DOTNET_SHARED_STORE = Join-Path $env:INSTALL_DIR "store"
$env:DOTNET_STARTUP_HOOKS = Join-Path $env:INSTALL_DIR "net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
$env:OTEL_DOTNET_AUTO_HOME = $env:INSTALL_DIR
$env:OTEL_DOTNET_AUTO_PLUGINS =
 "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

$env:OTEL_TRACES_EXPORTER=none
$env:OTEL_AWS_SIG_V4_ENABLED=true

$env:OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
$env:OTEL_METRICS_EXPORTER=none
$env:OTEL_LOGS_EXPORTER=none
$env:OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=http/protobuf
$env:OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=https://xray.us-east-1.amazonaws.com/v1/
traces
```

1. Start your application with the environment variables listed in the previous step.

2. (Optional) Alternatively, you can use the installation scripts provided to help installation and setup of AWS Distro for OpenTelemetry .NET auto-instrumentation package.

   For Linux, download and install the Bash installation script from the GitHub releases page:

   ```
   # Download and Install
   curl -L -O https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
   releases/latest/download/aws-otel-dotnet-install.sh
   chmod +x ./aws-otel-dotnet-install.sh
   ./aws-otel-dotnet-install.sh
   # Instrument
   . $HOME/.otel-dotnet-auto/instrument.shexport
    OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
   ```

   For Windows Server, download and install the PowerShell installation script from the GitHub releases page:

   ```
   # Download and Install
   $module_url = "https://github.com/aws-observability/aws-otel-dotnet-
   instrumentation/releases/latest/download/AWS.Otel.DotNet.Auto.psm1"
   $download_path = Join-Path $env:temp "AWS.Otel.DotNet.Auto.psm1"
   Invoke-WebRequest -Uri $module_url -OutFile $download_path
   Import-Module $download_path
   Install-OpenTelemetryCore
   # Instrument
   Import-Module $download_path
   Register-OpenTelemetryForCurrentSession -OTelServiceName "dotnet-service-name"
   Register-OpenTelemetryForIIS
   ```

   You can find the NuGet package of the AWS Distro for OpenTelemetry .NET auto-instrumentation package in the official NuGet repository. Be sure to check the README file for instructions.

Managed Prometheus collectors

Amazon CloudWatch managed Prometheus collectors provide a fully managed, agentless way to collect Prometheus-compatible metrics from your AWS resources and deliver them to the CloudWatch OTLP endpoint. You don't install, patch, or scale any collection infrastructure yourself, and you query the collected metrics with PromQL.

For more information, including how to integrate Amazon MSK, Amazon EKS, Amazon OpenSearch Service, and other VPC-based sources, see Amazon CloudWatch managed Prometheus collectors.

Setting up bearer token authentication for Metrics

Note

This page covers bearer token authentication for the CloudWatch Metrics OTLP endpoint. For CloudWatch Logs bearer token authentication, see Setting up bearer token authentication for Logs in the *CloudWatch Logs User Guide*.

Before you can send metrics using bearer token authentication with the CloudWatch OTLP endpoint, you need to:

- Create an IAM user with CloudWatch Metrics permissions
- Generate service-specific credentials (API key)

Important

We recommend using SigV4 authentication with short-term credentials for all workloads where this is possible. SigV4 provides the strongest security posture. Restrict the use of API keys (bearer tokens) to scenarios where short-term credential-based authentication is not feasible, such as sending metrics from non-AWS environments, third-party vendors, or platforms that do not support the AWS SDK. When you are ready to incorporate CloudWatch Metrics into applications with greater security requirements, switch to short-term credentials. For more information, see Alternatives to long-term access keys in the *IAM User Guide*.

Important

The CloudWatch OTLP endpoint requires TLS (HTTPS). Bearer token requests sent over plain HTTP are rejected. Always use `https://monitoring.`*`AWS Region`*`.amazonaws.com/v1/metrics` when configuring your client.

Option 1: Quick start using the AWS console

The AWS Management Console provides a streamlined workflow to generate API keys for OTLP endpoint access.

To set up OTLP endpoint access using the console

1. Sign in to the AWS Management Console.
2. Navigate to **CloudWatch** > **Settings** > **Global**.
3. In the API Keys section, choose **Generate API key**.
4. For **API key expiration**, do one of the following:
   - Select an API key expiration duration of **1**, **5**, **30**, **90**, or **365** days.
   - Choose **Custom duration** to specify a custom API key expiration date.
   - Select **Never expires** (not recommended).
5. Choose **Generate API key**.

The console automatically:

- Creates a new IAM user with appropriate permissions
- Attaches the CloudWatchAPIKeyAccess managed policy (includes `cloudwatch:PutMetricData` and `cloudwatch:CallWithBearerToken` permissions)
- Generates service-specific credentials (API key)

To save and verify your API key

1. Copy and securely save the displayed credentials:
   - **API Key ID** (Service-specific credential ID)
   - **API Key Secret** (Bearer token)

   The console also offers the option to store your API key directly in AWS Secrets Manager during generation. If you choose to store in Secrets Manager, the key is automatically updated on reset and deleted on key deletion.

> **Important**
>
> Save the API Key Secret immediately. You can't retrieve it later. If you lose it, you must generate a new API key.

2. Send a test metric to verify your setup:

```
curl -X POST "https://monitoring.us-east-1.amazonaws.com/v1/metrics" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer YOUR_API_KEY" \
     -d '{"resourceMetrics":[]}'
```

Option 2: Manual setup

If you prefer more control over the IAM configuration or need to customize permissions, you can set up the OTLP endpoint access manually.

Step 1: Create an IAM user

Create an IAM user for metric ingestion:

To create an IAM user for metric ingestion

1. Sign in to the AWS Management Console and navigate to IAM.
2. In the left navigation pane, choose **Users**.
3. Choose **Create user**.
4. Enter a user name (for example, **`cloudwatch-metrics-api-key-user`**).
5. Choose **Next**.
6. Attach one of the following IAM policies:

   Option A: Use the managed policy (recommended)

   Attach the CloudWatchAPIKeyAccess managed policy.

   Option B: Create a custom policy

   Create and attach the following IAM policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchMetricsAPIs",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:CallWithBearerToken",
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        },
        {
            "Sid": "KMSDecryptForCMKDatasets",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "cloudwatch.*.amazonaws.com",
                    "kms:EncryptionContext:aws:cloudwatch:arn":
 "arn:aws:cloudwatch:*:*:dataset/*"
                }
            },
            "Resource": "arn:aws:kms:*:*:key/*"
        }
    ]
}
```

7. Choose **Next** and then choose **Create user**.

> **Note**
>
> The KMS permissions are required if you plan to send metrics to datasets that use customer-managed KMS keys (CMK). The conditions restrict KMS access to only keys used through the CloudWatch service for dataset resources.

Step 2: Generate service-specific credentials (API key)

Generate the CloudWatch Metrics API key using the CreateServiceSpecificCredential API. You can also use the create-service-specific-credential AWS CLI command. For the credential age, you can specify a value between 1–36600 days. If you don't specify a credential age, the API key won't expire.

To generate an API key with an expiration of 30 days:

```
aws iam create-service-specific-credential \
    --user-name cloudwatch-metrics-api-key-user \
    --service-name cloudwatch.amazonaws.com \
    --credential-age-days 30
```

The response is a ServiceSpecificCredential object. The `ServiceCredentialSecret` value is your CloudWatch Metrics API key (bearer token).

> **Important**
>
> Store the `ServiceCredentialSecret` value securely. You can't retrieve it later. If you lose it, you must generate a new API key.

Step 3: Send metrics

You can immediately send metrics to the OTLP endpoint using your bearer token:

```
curl -X POST "https://monitoring.us-east-1.amazonaws.com/v1/metrics" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer YOUR_API_KEY" \
     -d '{"resourceMetrics":[]}'
```

The endpoint accepts both `application/json` and `application/x-protobuf` content types.

Control permissions for generating and using CloudWatch Metrics API keys

Controlling the generation of CloudWatch Metrics API keys

The `iam:CreateServiceSpecificCredential` action controls the generation of a service-specific key (such as a CloudWatch Metrics API key). You can scope this action to IAM users as a resource to limit the users for which a key can be generated.

You can use the following condition keys to impose conditions on the permission for the `iam:CreateServiceSpecificCredential` action:

- `iam:ServiceSpecificCredentialAgeDays` — Lets you specify, in the condition, the key's expiration time in days.
- `iam:ServiceSpecificCredentialServiceName` — Lets you specify, in the condition, the name of a service.

Controlling the usage of CloudWatch Metrics API keys

The `cloudwatch:CallWithBearerToken` action controls the use of a CloudWatch Metrics API key. To prevent an identity from using CloudWatch Metrics API keys, attach a policy that denies the `cloudwatch:CallWithBearerToken` action to the IAM user associated with the key.

> **Note**
>
> Bearer tokens for CloudWatch Metrics can only be used with the OTLP metrics ingestion endpoint (`https://monitoring.`*`AWS Region`*`.amazonaws.com/v1/metrics`). They cannot be used to call any other CloudWatch API or endpoint, including query APIs (`GetMetricData`, `ListMetrics`, `DescribeAlarms`), the PromQL query endpoint, the OTLP traces endpoint, or the OTLP logs endpoint.

Example policies

Prevent an identity from generating and using CloudWatch Metrics API keys:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyCWMetricsAPIKeys",
            "Effect": "Deny",
            "Action": [
                "iam:CreateServiceSpecificCredential",
                "cloudwatch:CallWithBearerToken"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

> **Warning**
>
> This policy prevents the creation of credentials for all AWS services that support creating service-specific credentials. For more information, see Service-specific credentials for IAM users in the *IAM User Guide*.

Prevent an identity from using CloudWatch Metrics API keys:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "cloudwatch:CallWithBearerToken",
            "Resource": "*"
        }
    ]
}
```

Allow the creation of CloudWatch Metrics keys only if they expire within 90 days:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceSpecificCredential",
            "Resource": "arn:aws:iam::123456789012:user/username",
            "Condition": {
                "StringEquals": {
                    "iam:ServiceSpecificCredentialServiceName":
 "cloudwatch.amazonaws.com"
                },
                "NumericLessThanEquals": {
                    "iam:ServiceSpecificCredentialAgeDays": "90"
                }
            }
        }
    ]
```

```
}
```

Using bearer tokens with the OpenTelemetry Collector

When using bearer tokens, you do not need the `sigv4auth` extension. Use the bearertokenauth extension to securely provide your API key from a file or environment variable:

```
extensions:
  bearertokenauth:
    filename: "/etc/otel/cw-api-key"

exporters:
  otlphttp:
    tls:
      insecure: false
    endpoint: https://monitoring.us-east-1.amazonaws.com/v1/metrics
    auth:
      authenticator: bearertokenauth

receivers:
  otlp:
    protocols:
      http:
        endpoint: 0.0.0.0:4318

processors:
  batch:
    send_batch_size: 200
    timeout: 10s

service:
  extensions: [bearertokenauth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]
```

Alternatively, you can reference an environment variable instead of a file:

```
extensions:
  bearertokenauth:
```

```
    token: "${env:CW_API_KEY}"
```

Important

Never hardcode API keys directly in collector configuration files. Configuration files are often committed to version control or stored in deployment manifests. Use `filename` to read from a mounted secret, or `${env:VAR}` to read from an environment variable injected by your secrets manager.

Note

With bearer token authentication, you do not need the `sigv4auth` extension, AWS credentials files, IAM roles, or IRSA configuration. This makes the collector configuration portable across any environment — AWS, on-premises, or other cloud providers.

Rotating API keys

Regularly rotating your API keys reduces the risk of unauthorized access. You should recommend establishing a rotation schedule that aligns with your organization's security policies.

Rotation process

To rotate an API key without interrupting metric delivery, follow this procedure:

To rotate an API key

1. Create a new (secondary) credential for the IAM user:

```
aws iam create-service-specific-credential \
    --user-name cloudwatch-metrics-api-key-user \
    --service-name cloudwatch.amazonaws.com \
    --credential-age-days 90
```

Note

IAM allows a maximum of 2 service-specific credentials per IAM user per service. Delete or deactivate old credentials before creating new ones if you have reached this limit.

2. (Optional) Store the new credential in AWS Secrets Manager for secure retrieval and automated rotation.
3. Update your OpenTelemetry Collector configuration or application to use the new API key.
4. Set the original credential to inactive:

```
aws iam update-service-specific-credential \
    --user-name cloudwatch-metrics-api-key-user \
    --service-specific-credential-id ACCA1234EXAMPLE1234 \
    --status Inactive
```

5. Verify that metric delivery is not impacted. Send a test request using the new key and confirm you receive an HTTP 200 response. You can also monitor your application's existing CloudWatch metrics to confirm data continues to arrive.
6. After confirming successful delivery with the new key, delete the previous credential:

```
aws iam delete-service-specific-credential \
    --service-specific-credential-id ACCA1234EXAMPLE1234
```

Monitoring key expiration

To check the creation date and status of your existing API keys, use the [list-service-specific-credentials] command:

```
aws iam list-service-specific-credentials \
    --user-name cloudwatch-metrics-api-key-user \
    --service-name cloudwatch.amazonaws.com
```

The response includes `CreateDate` and `Status` for each credential. Use this information to identify keys that are approaching expiration or have been active longer than your rotation policy allows.

Responding to a compromised API key

If you suspect that an API key has been compromised, take the following steps immediately:

To respond to a compromised API key

1. **Deactivate the key immediately** to prevent further unauthorized use:

```
aws iam update-service-specific-credential \
    --user-name cloudwatch-metrics-api-key-user \
    --service-specific-credential-id ACCA1234EXAMPLE1234 \
    --status Inactive
```

2. **Review CloudTrail logs** to determine the scope of unauthorized access. See the section called "Logging API key usage with CloudTrail" for how to enable auditing of API key usage.
3. **Create a replacement key** following the rotation process described in the section called "Rotation process".
4. **Delete the compromised key** after the replacement is in place:

```
aws iam delete-service-specific-credential \
    --service-specific-credential-id ACCA1234EXAMPLE1234
```

5. **Attach a deny policy** if you need to immediately block all bearer token access for the IAM user while you investigate:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "cloudwatch:CallWithBearerToken",
        "Resource": "*"
    }
}
```

Note

To carry out these actions through the API, you must authenticate with AWS credentials and not with a CloudWatch Metrics API key. Bearer tokens can only be used for metric ingestion, not for IAM management operations.

You can also use the following IAM API operations to manage compromised keys:

- ResetServiceSpecificCredential — Reset the key to generate a new password without deleting the credential. The key must not have expired.

Security best practices for API keys

Follow these best practices to protect your CloudWatch Metrics API keys:

- **Never embed API keys in source code.** Do not hard-code API keys in application code, collector configuration files, or version control systems. Use the `bearertokenauth` extension with `filename` or `${env:VAR}` to inject secrets at runtime.
- **Use a secrets manager.** Store API keys in AWS Secrets Manager or an equivalent secrets management solution. This enables centralized access control, audit logging, and automated rotation.
- **Set an expiration on all keys.** Always specify a `--credential-age-days` value when creating API keys. To enforce a maximum key lifetime across your organization, use the `iam:ServiceSpecificCredentialAgeDays` IAM condition key.
- **Apply least-privilege permissions.** Use the managed CloudWatchAPIKeyAccess policy as a starting point and restrict further as needed.
- **Enable CloudTrail logging.** Audit API key usage by enabling CloudTrail data events for `AWS::CloudWatch::Dataset`. See the section called “Logging API key usage with CloudTrail”.
- **Monitor with IAM Access Analyzer.** Use IAM Access Analyzer to identify unused credentials and overly permissive policies associated with your API key IAM users.
- **Rotate keys regularly.** Establish a rotation schedule and follow the process described in the section called “Rotating API keys”.

Logging API key usage with CloudTrail

You can use AWS CloudTrail to log data events for CloudWatch Metrics OTLP ingestion. CloudWatch emits `AWS::CloudWatch::Dataset` data events for calls to the OTLP endpoint, enabling you to audit metric ingestion activity including API key usage.

> **Note**
>
> The S3 bucket that you specify for the trail must have a bucket policy that allows CloudTrail to write log files to it. For more information, see Amazon S3 bucket policy for CloudTrail in the *AWS CloudTrail User Guide*.

To enable CloudTrail logging for CloudWatch Metrics API key usage

1. Create a trail:

```
aws cloudtrail create-trail \
    --name cloudwatch-metrics-api-key-audit \
    --s3-bucket-name my-cloudtrail-bucket \
    --region us-east-1
```

2. Configure advanced event selectors to capture CloudWatch Metrics write (ingestion) data events:

```
aws cloudtrail put-event-selectors \
    --region us-east-1 \
    --trail-name cloudwatch-metrics-api-key-audit \
    --advanced-event-selectors '[{
        "Name": "CloudWatch Metrics write data events",
        "FieldSelectors": [
            { "Field": "eventCategory", "Equals": ["Data"] },
            { "Field": "resources.type", "Equals": ["AWS::CloudWatch::Dataset"] },
            { "Field": "readOnly", "Equals": ["false"] }
        ]
    }]'
```

3. Start trail logging:

```
aws cloudtrail start-logging \
    --name cloudwatch-metrics-api-key-audit \
    --region us-east-1
```

The `readOnly: false` filter limits logging to write operations (PutMetricData), which includes all OTLP ingestion calls. To identify bearer token usage among these events, query your trail logs (through Athena or CloudTrail Lake) and filter by the IAM user name associated with your API key (for example, `cloudwatch-metrics-api-key-user`). Events from OTLP ingestion include `AdditionalEventData.protocol` set to `OTLP` in the event payload, which you can use in post-hoc queries to distinguish them from classic PutMetricData SDK calls.

Troubleshooting

The following are the common troubleshooting scenarios and solutions for OTLP endpoint.

Issue	Description	Solution
Non-existing AWS credentials when launching OCB collector	Collector throws the following error when starting. *Error: invalid configuration: extensions::sigv4auth: could not retrieve credential provider: failed to refresh cached credentials, no EC2 IMDS role found, operation error ec2imds: GetMetadata, request canceled, context deadline exceeded*.	Enter the correct credentials.
Invalid AWS credentials	Collector throws HTTP Status Code 403, Message=The security token included in the request is invalid., Details=[]" when sending requests though OTLP endpoint.	Refresh the AWS credentials on the collector server.
Transactions Search disabled	Collector throws Message=The OTLP API is supported with CloudWatch Logs as a Trace Segment Destination.	Make sure Transaction Search is enabled in CloudWatch before using the OTLP endpoint for traces. For more information, see Transaction Search.

Issue	Description	Solution
Batching and timeout issues	Collector throws one of these issues: • max elapsed time expired failed to make an HTTP request • io.opentelemetry.exporter.internal.http.HttpExporter - Failed to export spans. The request could not be executed. Full error message: timeout • io.opentelemetry.exporter.internal.grpc.GrpcExporter - Failed to export spans. Server responded with gRPC status code 2. Error message: timeout • rpc error: code = DeadlineExceeded desc = context deadline exceeded • rpc error: code = ResourceExhausted desc = Too many requests", "dropped_items": 1024	Tune batching and timeout policies using batchprocessor.

Issue	Description	Solution
Retry issues	Transient network issues between the collector and OTLP endpoint. • rpc error: code = Unavailable desc = error reading from server: read tcp • rpc error: code = Unavailable desc = unexpected HTTP status code received from server: 502 (Bad Gateway); • rpc error: code = Unavailable desc = unexpected HTTP status code received from server: 503 (Service Unavailable)	Tune retry policy using exporter.
Payload rejected	NA	Make sure the payload sent to the trace endpoint is within the limits and restrictions. For more information, see Endpoint limits and restrictions.

Issue	Description	Solution
No auth header injected to outgoing export requests in ADOT	Generic 403 error, "Missing Authentication Token": Example: `ERROR:opentelemetry.exporter.otlp.proto.http.trace_exporter:Failed to export batch code: 403, reason: Missing Authentication Token`	• Verify that you're using the correct supported version of ADOT. • Make sure your credentials are set up correctly by following the standard AWS SDK credential provider chain
Logs or spans not appearing in CloudWatch log groups	No logs or spans are appearing in the expected CloudWatch log groups (`aws/spans` for spans and your specified custom log group for logs).	Make sure that the library to be instrumented is supported by auto-instrumentation. See supported libraries for Java, Python, JavaScript and .NET.

Collect metrics, logs, and traces using the CloudWatch agent

The CloudWatch agent is a software component that collects metrics, logs, and traces from your Amazon EC2 instances, on-premises servers, and containerized applications. It enables you to monitor your infrastructure and applications more comprehensively than the basic monitoring provided by default.

The CloudWatch agent enables you to do the following:

- Collect internal system-level metrics from Amazon EC2 instances across operating systems. The metrics can include in-guest metrics, in addition to the metrics for Amazon EC2 instances. For a list of additional metrics that you can collect, see Metrics collected by the CloudWatch agent.
- Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by AWS.
- Retrieve custom metrics from your applications or services using the `StatsD` and `collectd` protocols. `StatsD` is supported on both Linux servers and servers running Windows Server. `collectd` is supported only on Linux servers.
- Collect logs from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server.

  > **Note**
  >
  > The CloudWatch agent does not support collecting logs from FIFO pipes.

- Send the metrics to either CloudWatch or Amazon Managed Service for Prometheus, or to both. The CloudWatch agent configuration file contains a `metrics_destinations` parameter in the `metrics` section. You can specify `cloudwatch`, `amp`, or both in this parameter.
- Version 1.300031.0 and later can be used to enable CloudWatch Application Signals. For more information, see Application Signals.
- Version 1.300025.0 and later can collect traces from OpenTelemetry or X-Ray client SDKs, and send them to X-Ray.

  Using the CloudWatch agent allows you to collect traces without needing to run a separate trace collection daemon, helping to reduce the number of agents that you run and manage.

Metrics that the CloudWatch agent publishes as CloudWatch metrics can be viewed in CloudWatch just as any other CloudWatch metrics. The default CloudWatch namespace for these metrics is `CWAgent`, although you can specify a different namespace when you configure the agent.

Metrics that the CloudWatch agent forwards to the CloudWatch OTLP endpoints are stored as OpenTelemetry metrics. These metrics don't use a CloudWatch namespace. Instead, you identify them by metric name and labels, and you query them with PromQL. For more information about OpenTelemetry metrics, see OpenTelemetry Metrics (Recommended). For information about querying these metrics, see Query metrics with PromQL.

The logs collected by the CloudWatch agent are processed and stored in Amazon CloudWatch Logs, just like logs collected by the older CloudWatch Logs agent. For information about CloudWatch Logs pricing, see Amazon CloudWatch Pricing.

Metrics collected by the CloudWatch agent are billed as custom metrics. For more information about CloudWatch metrics pricing, see Amazon CloudWatch Pricing.

The CloudWatch agent is open-source under the MIT license, and is hosted on GitHub. If you would like to build, customize or contribute to the CloudWatch agent, see the GitHub repository for the latest instructions. If you think you've found a potential security issue, do not post it on GitHub or any public forum. Instead, follow the instructions at Vulnerability Reporting or email AWS security directly.

For information about forwarding application metrics, traces, and logs through the CloudWatch agent using the OpenTelemetry Protocol (OTLP), see Example: send application metrics through the CloudWatch agent.

You can download and install the CloudWatch agent manually using the command line, or you can integrate it with AWS Systems Manager. The general flow of installing the CloudWatch agent is as follows:

1. Create IAM roles or users that enable the agent to collect metrics from the server and optionally to integrate with AWS Systems Manager.
2. Download the agent package.
3. Modify the CloudWatch agent configuration file and specify the metrics that you want to collect.
4. Install and start the agent on your servers.

Contents

- Supported operating systems
- Prerequisites
- Download the CloudWatch agent package
- Verifying the signature of the CloudWatch agent package
- Installing the CloudWatch agent
- Set up the CloudWatch agent with security-enhanced Linux (SELinux)
- Create the CloudWatch agent configuration file
- Starting the CloudWatch agent

- Metrics collected by the CloudWatch agent
- Using the CloudWatch agent with related telemetry
- Common scenarios with the CloudWatch agent
- CloudWatch agent credentials preference
- Troubleshooting the CloudWatch agent

Supported operating systems

The CloudWatch agent is supported on the following operating systems:

x86-64 architecture

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server versions 26.04, 25.04, 24.04, and 22.04
- Red Hat Enterprise Linux (RHEL) versions 10, 9, and 8
- Debian version 12
- SUSE Linux Enterprise Server (SLES) versions 16 and 15
- Oracle Linux versions 9 and 8
- AlmaLinux versions 10, 9, and 8
- Rocky Linux versions 10, 9, and 8
- macOS computers: EC2 M1 Mac1 instances, and computers running macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)
- Windows Server 2025, Windows Server 2022, Windows Server 2019, and Windows Server 2016
- Windows 11
- 64-bit Windows 10

ARM64 architecture

- Amazon Linux 2023
- Amazon Linux 2
- Ubuntu Server version 22.04

- Red Hat Enterprise Linux (RHEL) versions 9 and 8
- Debian versions 13 and 12
- SUSE Linux Enterprise Server (SLES) versions 16 and 15
- macOS computers: macOS 14 (Sonoma), macOS 13 (Ventura), and macOS 12 (Monterey)

Prerequisites

Make sure the following prerequisites are completed before installing the CloudWatch agent for the first time.

IAM roles and users for CloudWatch agent

Access to AWS resources requires permissions. You create an IAM role, an IAM user, or both to grant permissions that the CloudWatch agent needs to write metrics to CloudWatch.

Create an IAM role for Amazon EC2 instances

If you're going to run the CloudWatch agent on Amazon EC2 instances, create an IAM role with the necessary permissions.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Under **Trusted entity type**, verify that **AWS service** is selected. For **Use case**, choose **EC2**. Choose **Next**.
5. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy.
6. Choose **Next**.
7. For **Role name**, enter a name for the role, such as `CloudWatchAgentServerRole`. Choose **Create role**.

(Optional) If the agent is going to send logs to CloudWatch Logs and you want the agent to be able to set retention policies for these log groups, you need to add the `logs:PutRetentionPolicy` permission to the role.

Create IAM user for on-premises servers

If you're going to run the CloudWatch agent on on-premises servers, create an IAM user with the necessary permissions.

> **Note**
>
> This scenario requires IAM users with programmatic access and long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and then **Add users**.
3. Enter the user name for the new user. Select **Access key - Programmatic access** and choose **Next: Permissions**.
4. Choose **Attach existing policies directly**.
5. In the list of policies, select the check box next to **CloudWatchAgentServerPolicy**. If necessary, use the search box to find the policy. Choose **Next: Tags**.
6. (Optional) Create tags for the new IAM user. Choose **Next: Review**. Confirm that the correct policy is listed and choose **Create user**.
7. Next to the name of the new user, choose **Show**. Copy the access key and secret key to a file so that you can use them when installing the agent. Choose **Close**.

Attach an IAM role to an Amazon EC2 instance

To enable the CloudWatch agent to send data from an Amazon EC2 instance, you must attach the IAM role you created to the instance.

For more information about attaching an IAM role to an instance, see Attaching an IAM Role to an Instance in the Amazon Elastic Compute Cloud User Guide.

Allowing the CloudWatch agent to set log retention policy

You can configure the CloudWatch agent to set the retention policy for log groups that it sends log events to. If you do this, you must grant the `logs:PutRetentionPolicy` to the IAM role or user

that the agent uses. The agent uses an IAM role to run on Amazon EC2 instances, and uses an IAM user for on-premises servers.

To grant log retention permissions to the agent's IAM role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.
3. In the search box, enter the beginning of the name of the CloudWatch agent's IAM role. You chose this name when you created the role. It might be named `CloudWatchAgentServerRole`.

   When you see the role, choose the name of the role.
4. In the **Permissions** tab, choose **Add permissions**, **Create inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "logs:PutRetentionPolicy",
         "Resource": "*"
       }
     ]
   }
   ```
6. Choose **Review policy**.
7. For **Name**, enter **`CloudWatchAgentPutLogsRetention`** or something similar, and choose **Create policy**.

To grant log retention permissions to the agent's IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the left navigation pane, choose **Users**.
3. In the search box, enter the beginning of the name of the CloudWatch agent's IAM user. You chose this name when you created the user.

   When you see the user, choose the name of the user.
4. In the **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab and copy the following policy into the box, replacing the default JSON in the box:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:PutRetentionPolicy",
      "Resource": "*"
    }
  ]
}
```

6. Choose **Review policy**.
7. For **Name**, enter **`CloudWatchAgentPutLogsRetention`** or something similar, and choose **Create policy**.

Network requirements

> **Note**
>
> When the server is in a public subnet make sure there is access to an internet gateway. When the server is in a private subnet, access is through the NAT gateways or VPC Endpoint. For more information about NAT gateways, see https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html.

Your Amazon EC2 instances must have outbound internet access to send data to CloudWatch or CloudWatch Logs. For more information about how to configure internet access, see Internet Gateways in the Amazon VPC User Guide.

Using VPC endpoints

If you're using a VPC and want to use CloudWatch agent without public internet access, you can configure VPC endpoints for CloudWatch and CloudWatch Logs.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must add the CloudWatch endpoints for the appropriate Regions to the allow list. These endpoints are listed in Amazon CloudWatch endpoints and quotas.
- If you're using the agent to collect logs, you must add the CloudWatch Logs endpoints for the appropriate Regions to the allow list. These endpoints are listed in Amazon CloudWatch Logs endpoints and quotas.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must add the Systems Manager endpoints for the appropriate Regions to the allow list. These endpoints are listed in AWS Systems Manager endpoints and quotas.

Download the CloudWatch agent package

> **Note**
>
> To download the CloudWatch agent, your connection must use TLS 1.2 or later.

The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using this operating system, you can install the package by entering the following command. You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached.

```
sudo yum install amazon-cloudwatch-agent
```

On all supported operating systems, you can download and install the CloudWatch agent using the command line.

For each download link, there is a general link as well as links for each AWS Region.

If you use the Region-specific links, replace the default Region (*us-east-1*) with the appropriate Region for your account. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid download links are:

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is available for download. These files are in the following locations:

> **Note**
>
> If you use the Region-specific links, replace the default Region (*us-east-1*) with the appropriate Region for your account.

- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES` or `https://amazoncloudwatch-agent-`*us-east-1*`.s3.`*us-east-1*`.amazonaws.com/info/latest/RELEASE_NOTES`
- `https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION` or `https://amazoncloudwatch-agent-`*us-east-1*`.s3.`*us-east-1*`.amazonaws.com/info/latest/CWAGENT_VERSION`

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/

Architecture	Platform	Download link	Signature file link
	and Amazon Linux 2	amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
ARM64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/arm64/latest/amazon-cloudwatch-agent.pkg.sig

Verifying the signature of the CloudWatch agent package

GPG signature files are included for CloudWatch agent packages on Linux servers. You can use a public key to verify the agent download file is original and unmodified.

For Windows Server, you can use the MSI to verify the signature. For macOS computers, the signature is included in the agent download package.

To find the correct signature file, use the following table. For each architecture and operating system, you can see a general link and links for each Region.

If you use the Region-specific links, replace the default Region (*us-east-1*) with the appropriate Region for your account. For example, for Amazon Linux 2023 and Amazon Linux 2 and the x86-64 architecture, three of the valid links are:

- `https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig`
- `https://amazoncloudwatch-agent-us-east-1.s3.us-east-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`
- `https://amazoncloudwatch-agent-eu-central-1.s3.eu-central-1.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm`

Note

To download the CloudWatch agent, your connection must use TLS 1.2 or later.

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	CentOS	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
		https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/centos/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/amd64/latest/amazon-cloudwatch-agent.rpm.sig
x86-64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.sig
x86-64	Oracle	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/oracle_linux/amd64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	macOS	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg	https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg.sig
x86-64	Windows	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi	https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi.sig
ARM64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
ARM64	Red Hat	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/redhat/arm64/latest/amazon-cloudwatch-agent.rpm.sig
ARM64	Ubuntu	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/ubuntu/arm64/latest/amazon-cloudwatch-agent.deb.sig
ARM64	Debian	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb	https://amazoncloudwatch-agent.s3.amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/debian/arm64/latest/amazon-cloudwatch-agent.deb.sig

Architecture	Platform	Download link	Signature file link
ARM64	SUSE	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm	https://amazoncloudwatch-agent.s3.amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig https://amazoncloudwatch-agent-*us-east-1* .s3.*us-east-1* .amazonaws.com/suse/arm64/latest/amazon-cloudwatch-agent.rpm.sig

To verify the CloudWatch agent package on a Linux server

1. Download the public key.

```
shell$ wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-
cloudwatch-agent.gpg
```

2. Import the public key into your keyring.

```
shell$  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

   Make a note of the key value, as you need it in the next step. In the preceding example, the key value is 3B789C72.

3. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
shell$  gpg --fingerprint key-value
pub   2048R/3B789C72 2017-11-14
      Key fingerprint = 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid                  Amazon CloudWatch Agent
```

   The fingerprint string should be equal to the following:

```
9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72
```

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

4. Download the package signature file using **wget**. To determine the correct signature file, see the preceding table.

```
wget Signature File Link
```

5. To verify the signature, run **gpg --verify**.

```
shell$ gpg --verify signature-filename agent-download-filename
gpg: Signature made Wed 29 Nov 2017 03:00:59 PM PST using RSA key ID 3B789C72
gpg: Good signature from "Amazon CloudWatch Agent"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

If the output includes the phrase BAD `signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a server running Windows Server

1. Download and install GnuPG for Windows from https://gnupg.org/download/. When installing, include the **Shell Extension (GpgEx)** option.

   You can perform the remaining steps in Windows PowerShell.

2. Download the public key.

```
PS> wget https://amazoncloudwatch-agent.s3.amazonaws.com/assets/amazon-cloudwatch-
agent.gpg -OutFile amazon-cloudwatch-agent.gpg
```

3. Import the public key into your keyring.

```
PS>  gpg --import amazon-cloudwatch-agent.gpg
gpg: key 3B789C72: public key "Amazon CloudWatch Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

   Make a note of the key value because you need it in the next step. In the preceding example, the key value is 3B789C72.

4. Verify the fingerprint by running the following command, replacing *key-value* with the value from the preceding step:

```
PS>  gpg --fingerprint key-value
pub   rsa2048 2017-11-14 [SC]
      9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
uid           [ unknown] Amazon CloudWatch Agent
```

   The fingerprint string should be equal to the following:

   `9376 16F3 450B 7D80 6CBD 9725 D581 6730 3B78 9C72`

   If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

   After you have verified the fingerprint, you can use it to verify the signature of the CloudWatch agent package.

5. Download the package signature file using wget. To determine the correct signature file, see CloudWatch Agent Download Links.

6. To verify the signature, run **gpg --verify**.

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
gpg:                using RSA key D58167303B789C72
gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
```

If the output includes the phrase `BAD signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

To verify the CloudWatch agent package on a macOS computer

- There are two methods for signature verification on macOS.
  - Verify the fingerprint by running the following command.

```
pkgutil --check-signature amazon-cloudwatch-agent.pkg
```

You should see a result similar to the following.

```
Package "amazon-cloudwatch-agent.pkg":
        Status: signed by a developer certificate issued by Apple for
 distribution
        Signed with a trusted timestamp on: 2020-10-02 18:13:24 +0000
        Certificate Chain:
        1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
        Expires: 2024-10-18 22:31:30 +0000
        SHA256 Fingerprint:
        81 B4 6F AF 1C CA E1 E8 3C 6F FB 9E 52 5E 84 02 6E 7F 17 21 8E FB
        0C 40 79 13 66 8D 9F 1F 10 1C

 ------------------------------------------------------------------------
        2. Developer ID Certification Authority
        Expires: 2027-02-01 22:12:15 +0000
        SHA256 Fingerprint:
        7A FC 9D 01 A6 2F 03 A2 DE 96 37 93 6D 4A FE 68 09 0D 2D E1 8D 03
        F2 9C 88 CF B0 B1 BA 63 58 7F

 ------------------------------------------------------------------------
        3. Apple Root CA
        Expires: 2035-02-09 21:40:36 +0000
        SHA256 Fingerprint:
        B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C
```

```
        68 C5 BE 91 B5 A1 10 01 F0 24
```

- Or, download and use the .sig file To use this method, follow these steps.
  - Install the GPG application to your macOS host by entering the following command.

    ```
    brew install GnuPG
    ```

- Download the package signature file using curl. To determine the correct signature file, see CloudWatch Agent Download Links.
- To verify the signature, run **gpg --verify**.

  ```
  PS> gpg --verify sig-filename agent-download-filename
  gpg: Signature made 11/29/17 23:00:45 Coordinated Universal Time
  gpg:                using RSA key D58167303B789C72
  gpg: Good signature from "Amazon CloudWatch Agent" [unknown]
  gpg: WARNING: This key is not certified with a trusted signature!
  gpg:          There is no indication that the signature belongs to the owner.
  Primary key fingerprint: 9376 16F3 450B 7D80 6CBD  9725 D581 6730 3B78 9C72
  ```

  If the output includes the phrase BAD `signature`, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

  Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Installing the CloudWatch agent

You can install the CloudWatch agent on your Amazon EC2 instances, on-premises servers, and in containerized environments to collect metrics, logs, and traces. The agent supports Linux, Windows, and macOS operating systems. There are several ways to install the agent, including using Systems Manager, from the CloudWatch console, installing from the command line, or installing using a configuration file. Before installing, make sure you have the required IAM permissions and network access configured.

Topics

- Install and Configure Amazon CloudWatch Agent with Workload Detection in the CloudWatch console
- Manual installation on Amazon EC2
- Install the CloudWatch agent using AWS Systems Manager
- Install the CloudWatch agent on on-premises servers
- Install the CloudWatch agent on Azure
- Install the CloudWatch agent on Google Cloud
- Install the CloudWatch agent on new instances using CloudFormation

Install and Configure Amazon CloudWatch Agent with Workload Detection in the CloudWatch console

Introduction

You can use the CloudWatch Getting Started console to install and configure the CloudWatch agent on Amazon EC2 instances. The Amazon CloudWatch agent is a lightweight software component that collects system-level metrics, logs, and traces from your Amazon EC2 instances. By automating the collection and delivery of monitoring data to CloudWatch, the agent enables you to gain actionable insights, optimize resource utilization, and make sure your applications are running smoothly with minimal configuration effort.

Configure the CloudWatch agent with pre-defined, workload-specific configurations that leverage automatic workload detection to identify running applications and services on your instances. You can customize data collection with specific metrics, logs, and traces, enabling you to monitor application performance and troubleshoot issues effectively.

How it Works

The CloudWatch agent detects workloads running on your Amazon EC2 instances through automatic workload detection capabilities. This feature identifies running applications and services on your instances, enabling intelligent monitoring without manual configuration.

Observability solutions provide pre-defined, workload-specific configurations tailored to common applications such as Apache Kafka, Apache Tomcat, Java Virtual Machines (JVM), NGINX, and NVIDIA GPU workloads. These solutions streamline the monitoring setup by automatically collecting the right metrics, logs, and traces specific to each detected workload, eliminating the need for manual instrumentation and configuration.

When workload detection is enabled, the agent analyzes your instance environment and automatically selects relevant pre-configured monitoring templates. These configurations are optimized by AWS subject-matter experts to capture the most important telemetry data for each workload type, ensuring you have comprehensive observability from the start.

Prerequisites

SSM Agent installation (required)

You must have AWS Systems Manager (SSM) agent installed on your Amazon EC2 instances. SSM agent comes pre-installed on most AWS-supplied Amazon Machine Images (AMIs), If you need to manually install or update the SSM agent, refer to the Systems Manager documentation.

> **Note**
>
> Default Host Management Configuration (DHMC) is a Systems Manager feature that automatically grants Amazon EC2 instances permissions to connect to Systems Manager without requiring you to manually attach an IAM instance profile to each instance. If your Amazon EC2 instances are using DHMC and the CloudWatch agent installation process attaches the CloudWatch policy to your instances, it can take up to 30 minutes for the new policy to take effect. This delay can postpone the publication of metrics, logs, and traces to CloudWatch. To mitigate, you can create your Amazon EC2 instances with an IAM role that contains the AmazonSSMManagedInstanceCore policy.

Workload Detection (Recommended)

Workload detection is an opt-in feature that automatically identifies running applications and services on your instances. We recommend turning on workload detection to take advantage of pre-configured, workload-specific monitoring templates. You can enable workload detection in the CloudWatch console settings.

Getting started

Open the Getting Started with Amazon CloudWatch agent page in the Amazon CloudWatch console: https://console.aws.amazon.com/cloudwatch/home#cloudwatch-agent

Manual instance deployment for CloudWatch Agent

Manually select up to 50 instances for CloudWatch agent installation and configuration. This targeted approach allows you to enhance monitoring for specific Amazon EC2 instances.

Tag-based deployment for CloudWatch Agent

Use tag-based deployment to install and configure the CloudWatch agent on fleets of Amazon EC2 instances. This method applies to all current and future instances with matching tags.

Tag-based configuration

Tag-based configurations allow you to efficiently organize, view, and modify configurations, helping you to manage CloudWatch agent and its configuration across fleets of Amazon EC2 instances.

CloudWatch agent installation

Install the CloudWatch agent to collect metrics, logs, and traces from Amazon EC2 instances and on-premises hosts. This telemetry provides important health and performance data about your infrastructure and applications.

CloudWatch agent configuration

Configure the CloudWatch agent with pre-defined, workload-specific configurations. You can customize data collection with specific metrics, logs, and traces, enabling you to monitor application performance and troubleshoot issues effectively.

Costs

Additional metrics that you add during this process are billed as custom metrics. For more information about CloudWatch metrics pricing, see Amazon CloudWatch Pricing.

Manual installation on Amazon EC2

> **Note**
>
> Make sure the prerequisites are completed before installing the CloudWatch agent for the first time.

Topics

- Installing on Amazon Linux using the package manager
- Installing on Amazon Linux using the command line
- Installing on Windows
- Installing on macOS

Installing on Amazon Linux using the package manager

The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using one of these operating systems, you can install the package by entering the following command:

```
sudo yum install amazon-cloudwatch-agent
```

You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached.

Installing on Amazon Linux using the command line

On all supported Linux operating systems, you can download and install the CloudWatch agent using the command line.

1. Download the CloudWatch agent. For a Linux server, enter the following command. For `download-link`, use the appropriate download link from the table below.

   ```
   wget download-link
   ```

Architecture	Platform	Download link
x86-64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
ARM64	Amazon Linux 2023 and Amazon Linux 2	https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/arm64/latest/amazon-cloudwatch-agent.rpm

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see Verifying the signature of the CloudWatch agent package.
3. Install the package. If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

```
sudo rpm -U ./amazon-cloudwatch-agent.rpm
```

If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

Installing on Windows

On Windows Server, you can download and install the CloudWatch agent using the command line.

1. Download the following file:

   ```
   https://amazoncloudwatch-agent.s3.amazonaws.com/windows/amd64/latest/amazon-cloudwatch-agent.msi
   ```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see Verifying the signature of the CloudWatch agent package.
3. Install the package. Change to the directory containing the package and enter the following:

   ```
   msiexec /i amazon-cloudwatch-agent.msi
   ```

   This command also works from within PowerShell. For more information about MSI command options, see Command-Line Options in the Microsoft Windows documentation.

Installing on macOS

On macOS computers, you can download and install the CloudWatch agent using the command line.

1. Download the appropriate package for your architecture:

   ```
   https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/amd64/latest/amazon-cloudwatch-agent.pkg
   ```

   For ARM64 architecture:

```
https://amazoncloudwatch-agent.s3.amazonaws.com/darwin/arm64/latest/amazon-
cloudwatch-agent.pkg
```

2. After you have downloaded the package, you can optionally verify the package signature. For more information, see Verifying the signature of the CloudWatch agent package.
3. Install the package. Change to the directory containing the package and enter the following:

```
sudo installer -pkg ./amazon-cloudwatch-agent.pkg -target /
```

Install the CloudWatch agent using AWS Systems Manager

Using AWS Systems Manager makes it easier to install the CloudWatch agent on a fleet of Amazon EC2 instances. You can download the agent into one server and create your CloudWatch agent configuration file for all servers in the fleet. Then you can use Systems Manager to install the agent on the other servers, using the configuration file that you created. Use the following topics to install and run the CloudWatch agent using AWS Systems Manager.

Topics

- Install or update the SSM Agent
- Verify Systems Manager prerequisites
- Verify internet access
- Download the CloudWatch agent package to your first instance
- Create and modify the agent configuration file
- Install and start the CloudWatch agent on additional EC2 instances using your agent configuration
- Install the CloudWatch agent on additional EC2 instances using your agent configuration
- (Optional) Modify the common configuration and named profile for CloudWatch agent

Install or update the SSM Agent

On an Amazon EC2 instance, the CloudWatch agent requires that the instance is running version 2.2.93.0 or later of the SSM Agent. Before you install the CloudWatch agent, update or install SSM Agent on the instance if you haven't already done so.

For information about installing or updating SSM Agent on an instance running Linux, see Installing and Configuring SSM Agent on Linux Instances in the *AWS Systems Manager User Guide*.

For information about installing or updating the SSM Agent, see Working with SSM Agent in the *AWS Systems Manager User Guide*.

Verify Systems Manager prerequisites

Before you use Systems Manager Run Command to install and configure the CloudWatch agent, verify that your instances meet the minimum Systems Manager requirements. For more information, see Systems Manager Prerequisites in the *AWS Systems Manager User Guide*.

Verify internet access

Your Amazon EC2 instances must be able to connect to CloudWatch endpoints. This can be by Internet Gateway, NAT gateway, or CloudWatch Interface VPC endpoints. For more information about how to configure internet access, see Internet Gateways in the *Amazon VPC User Guide*.

The endpoints and ports to configure on your proxy are as follows:

- If you're using the agent to collect metrics, you must allow list the CloudWatch endpoints for the appropriate Regions. These endpoints are listed in Amazon CloudWatch in the *Amazon Web Services General Reference*.
- If you're using the agent to collect logs, you must allow list the CloudWatch Logs endpoints for the appropriate Regions. These endpoints are listed in Amazon CloudWatch Logs in the *Amazon Web Services General Reference*.
- If you're using Systems Manager to install the agent or Parameter Store to store your configuration file, you must allow list the Systems Manager endpoints for the appropriate Regions. These endpoints are listed in AWS Systems Manager in the *Amazon Web Services General Reference*.

Download the CloudWatch agent package to your first instance

Use the following steps to download the CloudWatch agent package using Systems Manager.

To download the CloudWatch agent using Systems Manager

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

-or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance to install the CloudWatch agent on. If you don't see a specific instance, it might not be configured as a managed instance for use with Systems Manager. For more information, see Setting Up AWS Systems Manager for Hybrid Environments in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** field, enter *AmazonCloudWatchAgent*.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Create and modify the agent configuration file

After you have downloaded the CloudWatch agent, you must create the configuration file before you start the agent on any servers.

If you're going to save your agent configuration file in the Systems Manager Parameter Store, you must use an EC2 instance to save to the Parameter Store. Additionally, you must first attach to that instance the `CloudWatchAgentAdminRole` IAM role. For more information about attaching roles, see Attaching an IAM Role to an Instance in the *Amazon EC2 User Guide*.

For more information about creating the CloudWatch agent configuration file, see Create the CloudWatch agent configuration file.

Install and start the CloudWatch agent on additional EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

For each of these servers, follow the steps listed previously in this section to verify the Systems Manager prerequisites, the version of the SSM Agent, and internet access. Then use the following instructions to install the CloudWatch agent on the additional instances, using the CloudWatch agent configuration file that you have created.

Step 1: Download and install the CloudWatch agent

To be able to send the CloudWatch data to a different Region, make sure that the IAM role that you attached to this instance has permissions to write the CloudWatch data in that Region.

Following is an example of using the `aws configure` command to create a named profile for the CloudWatch agent. This example assumes that you are using the default profile name of `AmazonCloudWatchAgent`.

To create the AmazonCloudWatchAgent profile for the CloudWatch agent

- On Linux servers, type the following command and follow the prompts:

  ```
  sudo aws configure --profile AmazonCloudWatchAgent
  ```

  On Windows Server, open PowerShell as an administrator, type the following command and follow the prompts.

  ```
  aws configure --profile AmazonCloudWatchAgent
  ```

Install the CloudWatch agent on additional EC2 instances using your agent configuration

After you have a CloudWatch agent configuration saved in Parameter Store, you can use it when you install the agent on other servers.

For each of these servers, follow the steps listed previously in this section to verify the Systems Manager prerequisites, the version of the SSM Agent, and internet access. Then use the following instructions to install the CloudWatch agent on the additional instances, using the CloudWatch agent configuration file that you have created.

Step 1: Download and install the CloudWatch agent

You need to install the agent on each server where you will run the agent. The CloudWatch agent is available as a package in Amazon Linux 2023 and Amazon Linux 2. If you are using this operating system, you can use Systems Manager to install the package by using the following steps.

> **Note**
>
> You must also make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** attached. For more information, see Prerequisites.

To use Systems Manager to install the CloudWatch agent package

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select **AWS-RunShellScript**. Then paste the following into **Command parameters**.

   ```
   sudo yum install amazon-cloudwatch-agent
   ```
5. Choose **Run**

On all supported operating systems, you can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link.

> **Note**
>
> When you install or update the CloudWatch agent, only the **Uninstall and reinstall** option is supported. You can't use the **In-place update** option.

Systems Manager Run Command enables you to manage the configuration of your instances. You specify a Systems Manager document, specify parameters, and execute the command on one or more instances. SSM Agent on the instance processes the command and configures the instance as specified.

To download the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AWS-ConfigureAWSPackage**.
5. In the **Targets** area, choose the instance on which to install the CloudWatch agent. If you do not see a specific instance, it might not be configured for Run Command. For more information, see Setting Up AWS Systems Manager for Hybrid Environments in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter `AmazonCloudWatchAgent`.
8. Keep **Version** set to **latest** to install the latest version of the agent.
9. Choose **Run**.
10. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully installed.

Step 2: Start the CloudWatch agent using your agent configuration file

Follow these steps to start the agent using Systems Manager Run Command.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

To start the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.

3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **configure**.
7. In the **Optional Configuration Source** list, choose **ssm**.
8. In the **Optional Configuration Location** box, enter the name of the Systems Manager parameter name of the agent configuration file that you created and saved to Systems Manager Parameter Store, as explained in Create the CloudWatch agent configuration file.
9. In the **Optional Restart** list, choose **yes** to start the agent after you have finished these steps.
10. Choose **Run**.
11. Optionally, in the **Targets and outputs** areas, select the button next to an instance name and choose **View output**. Systems Manager should show that the agent was successfully started.

(Optional) Modify the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can use this file to optionally specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"
```

```
## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used.

  On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

  If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.
- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

  The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Install the CloudWatch agent on on-premises servers

If you downloaded the CloudWatch agent on a computer and created your agent configuration file, you can use that configuration file to install the agent in other on-premises servers.

Download the CloudWatch agent on an on-premises server

You can download the CloudWatch agent package using either Systems Manager Run Command or an Amazon S3 download link.

Download using Systems Manager

To use Systems Manager Run Command, you must register your on-premises server with Amazon EC2 Systems Manager. For more information, see Setting Up Systems Manager in Hybrid Environments in the *AWS Systems Manager User Guide*.

If you have already registered your server, update SSM Agent to the latest version.

For information about updating SSM Agent on a server running Linux, see Install SSM Agent for a Hybrid Environment (Linux) in the *AWS Systems Manager User Guide*.

For information about updating the SSM Agent on a server running Windows Server, see Install SSM Agent for a Hybrid Environment (Windows) in the *AWS Systems Manager User Guide*.

To use the SSM Agent to download the CloudWatch agent package on an on-premises server

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AWS-ConfigureAWSPackage**.

5. In the **Targets** area, select the server to install the CloudWatch agent on. If you don't see a specific server, it might not be configured for Run Command. For more information, see Setting Up AWS Systems Manager for Hybrid Environments in the *AWS Systems Manager User Guide*.
6. In the **Action** list, choose **Install**.
7. In the **Name** box, enter *AmazonCloudWatchAgent*.
8. Keep **Version** blank to install the latest version of the agent.
9. Choose **Run**.

   The agent package is downloaded, and the next steps are to configure and start it.

(Installing on an on-premises server) Specify IAM credentials and AWS Region

To enable the CloudWatch agent to send data from an on-premises server, you must specify the access key and secret key of the IAM user that you created earlier.

You also must specify the AWS Region to send the metrics to, using the `region` field.

Following is an example of this file.

```
[AmazonCloudWatchAgent]
aws_access_key_id=my_access_key
aws_secret_access_key=my_secret_key
region = us-west-1
```

For *my_access_key* and *my_secret_key*, use the keys from the IAM user that doesn't have the permissions to write to Systems Manager Parameter Store.

If you name this profile `AmazonCloudWatchAgent`, you don't need to do anything more. Optionally, you can give it a different name and specify that name as the value for `shared_credential_profile` in the `common-config.toml` file, which is explained in the following section.

Following is an example of using the **aws configure** command to create a named profile for the CloudWatch agent. This example assumes that you're using the default profile name of `AmazonCloudWatchAgent`.

To create the AmazonCloudWatchAgent profile for the CloudWatch agent

1. If you haven't already done so, install the AWS Command Line Interface on the server. For more information, see Installing the AWS CLI.
2. On Linux servers, enter the following command and follow the prompts:

   ```
   sudo aws configure --profile AmazonCloudWatchAgent
   ```

   On Windows Server, open PowerShell as an administrator, enter the following command, and follow the prompts.

   ```
   aws configure --profile AmazonCloudWatchAgent
   ```

(Optional) Modifying the common configuration and named profile for CloudWatch agent

The CloudWatch agent includes a configuration file called `common-config.toml`. You can optionally use this file to specify proxy and Region information.

On a server running Linux, this file is in the `/opt/aws/amazon-cloudwatch-agent/etc` directory. On a server running Windows Server, this file is in the `C:\ProgramData\Amazon\AmazonCloudWatchAgent` directory.

The default `common-config.toml` is as follows:

```
# This common-config is used to configure items used for both ssm and cloudwatch access


## Configuration for shared credential.
## Default credential strategy will be used if it is absent here:
##            Instance role is used for EC2 case by default.
##            AmazonCloudWatchAgent profile is used for onPremise case by default.
# [credentials]
#    shared_credential_profile = "{profile_name}"
#    shared_credential_file= "{file_name}"

## Configuration for proxy.
## System-wide environment-variable will be read if it is absent here.
## i.e. HTTP_PROXY/http_proxy; HTTPS_PROXY/https_proxy; NO_PROXY/no_proxy
## Note: system-wide environment-variable is not accessible when using ssm run-command.
## Absent in both here and environment-variable means no proxy will be used.
```

```
# [proxy]
#    http_proxy = "{http_url}"
#    https_proxy = "{https_url}"
#    no_proxy = "{domain}"
```

All lines are commented out initially. To set the credential profile or proxy settings, remove the # from that line and specify a value. You can edit this file manually, or by using the `RunShellScript` Run Command in Systems Manager:

- `shared_credential_profile` – For on-premises servers, this line specifies the IAM user credential profile to use to send data to CloudWatch. If you keep this line commented out, `AmazonCloudWatchAgent` is used. For more information about creating this profile, see (Installing on an on-premises server) Specify IAM credentials and AWS Region.

  On an EC2 instance, you can use this line to have the CloudWatch agent send data from this instance to CloudWatch in a different AWS Region. To do so, specify a named profile that includes a `region` field specifying the name of the Region to send to.

  If you specify a `shared_credential_profile`, you must also remove the # from the beginning of the `[credentials]` line.

- `shared_credential_file` – To have the agent look for credentials in a file located in a path other than the default path, specify that complete path and file name here. The default path is `/root/.aws` on Linux and is `C:\\Users\\Administrator\\.aws` on Windows Server.

  The first example below shows the syntax of a valid `shared_credential_file` line for Linux servers, and the second example is valid for Windows Server. On Windows Server, you must escape the \ characters.

  ```
  shared_credential_file= "/usr/username/credentials"
  ```

  ```
  shared_credential_file= "C:\\Documents and Settings\\username\\.aws\\credentials"
  ```

  If you specify a `shared_credential_file`, you must also remove the # from the beginning of the `[credentials]` line.

- Proxy settings – If your servers use HTTP or HTTPS proxies to contact AWS services, specify those proxies in the `http_proxy` and `https_proxy` fields. If there are URLs that should be excluded from proxying, specify them in the `no_proxy` field, separated by commas.

Install the CloudWatch agent on Azure

You can run the Amazon CloudWatch agent on Azure to collect metrics, logs, and traces. The agent sends this telemetry to Amazon CloudWatch in your AWS account. This page covers two environments:

- Azure Virtual Machines – You install the agent package directly on the machine.
- Azure Kubernetes Service (AKS) – You install the agent through the Amazon CloudWatch Observability Helm chart.

For each environment, you can either use the onboarding scripts or follow the manual steps.

> **Finding your Azure resource ID**
>
> The onboarding scripts identify your virtual machine or cluster by its full Azure resource ID (`CWAGENT_AZURE_RESOURCE_ID`). You can find the resource ID on the resource's Properties page in the Azure portal, or retrieve it by running `az vm show` (or `az aks show`) with `--query id --output tsv`.

How the agent authenticates from Azure

We recommend that you federate an Azure identity to an AWS IAM role so that the agent uses temporary credentials. This approach avoids storing long-lived AWS access keys on the virtual machine or cluster, which reduces your security risk. The exact mechanism depends on the environment:

- On a virtual machine, the Azure managed identity issues OpenID Connect (OIDC) tokens. The agent obtains a token from the Azure Instance Metadata Service and calls `AssumeRoleWithWebIdentity` to obtain temporary AWS credentials.
- On AKS, the cluster's workload identity issues the OIDC token through a projected service account token, and the agent assumes the role in the same way.

In both cases, your AWS account needs the following:

- An IAM OIDC identity provider for the Azure issuer
- An IAM role that trusts the issuer and has the CloudWatchAgentServerPolicy managed policy attached

Azure Virtual Machines

On an Azure virtual machine, you install the agent package in the same way as on an on-premises server. The virtual machine authenticates to AWS with a managed identity that is federated to an AWS IAM role.

Automated setup (recommended)

The CloudWatch agent provides onboarding scripts that automate the setup. The scripts assign the Azure managed identity, install the agent, and create the AWS IAM role and trust policy. The scripts configure the agent with a default OpenTelemetry configuration. This configuration collects host metrics (such as CPU, memory, disk, and network) and starts an OpenTelemetry Protocol (OTLP) receiver for metrics, logs, and traces from applications on the machine. The agent enriches all of this telemetry and forwards it to the CloudWatch OTLP endpoints.

Run the Azure step first, then the AWS trust step. Provide the role ARN that the AWS trust step creates or updates (by default, `arn:aws:iam::`*`account-id`*`:role/CloudWatchAgentServerRole`).

> **Reusing an existing IAM role**
>
> Both scripts are safe to run even if the IAM role already exists. The AWS trust step merges the Azure trust into the role's existing trust policy instead of replacing it, so other trust statements remain. It attaches `CloudWatchAgentServerPolicy` only if the role doesn't already have it, and leaves the role's other policies unchanged.

To set up the agent on an Azure virtual machine using the onboarding scripts

1. On a machine with the Azure CLI signed in (for example, Azure Cloud Shell), run the Azure step with the role ARN and the AWS Region. The script assigns the virtual machine's managed identity, installs and starts the agent, and prints the Azure tenant ID.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/azure/setup.sh | \
  CWAGENT_PLATFORM=azure_vm \
  CWAGENT_AZURE_RESOURCE_ID=vm-resource-id \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

2. On a machine with AWS credentials that have IAM write access to the target account (for example, AWS CloudShell), run the AWS trust step with the tenant ID from the previous step. The script creates the role, attaches `CloudWatchAgentServerPolicy`, and adds the Azure web-identity trust.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/aws/setup.sh | \
  CWAGENT_PLATFORM=azure_vm \
  CWAGENT_AZURE_TENANT_ID=tenant-id \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

Manual setup

To assign a managed identity to the virtual machine

> **Active Azure CLI subscription**
>
> The following `az` commands act on the subscription that is active in the Azure CLI. Confirm it with `az account show`, or switch with `az account set --subscription subscription-id`, before you run them.

1. Assign a system-assigned managed identity to the virtual machine.

```
az vm identity assign --resource-group resource-group --name vm-name
```

2. Retrieve the Azure tenant ID to use in the next procedure.

```
az account show --query tenantId --output tsv
```

To create the IAM role in your AWS account

1. Register the Microsoft Entra ID issuer as an IAM OIDC identity provider. The audience (`--client-id-list`) is `https://management.azure.com/`. You don't need to supply a thumbprint. IAM retrieves the top intermediate certificate authority (CA) thumbprint of the issuer's server certificate.

```
aws iam create-open-id-connect-provider \
  --url https://sts.windows.net/tenant-id/ \
  --client-id-list https://management.azure.com/
```

2. Create a trust policy that allows the Azure identity to assume the role. Save it to a file named `trust-policy.json`. Replace *account-id* with your AWS account ID and *tenant-id* with the Azure tenant ID.

> **Reusing an existing IAM role**
>
> If you reuse an existing role, merge this policy with the role's existing trust policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Federated": "arn:aws:iam::account-id:oidc-provider/
sts.windows.net/tenant-id/" },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": { "StringEquals": { "sts.windows.net/tenant-id/:aud": "https://
management.azure.com/" } }
    }
  ]
}
```

3. Create the role and attach the CloudWatchAgentServerPolicy managed policy. Note the role ARN that the first command returns.

```
aws iam create-role --role-name role-name --assume-role-policy-document file://
trust-policy.json
aws iam attach-role-policy --role-name role-name --policy-arn
 arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

To install and start the agent

1. Download and install the agent on the virtual machine, in the same way as on an on-premises server. See Install the CloudWatch agent on on-premises servers.

2. Set the AWS Region and the ARN of the role to assume, then start the agent with the default OpenTelemetry configuration (`default:otel`). The default OpenTelemetry configuration reads the role ARN from the `CWAGENT_ROLE_ARN` environment variable.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a set-env -e
 AWS_REGION=region
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a set-env -e
 CWAGENT_ROLE_ARN=role-arn
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m auto -c default:otel -s
```

Azure Kubernetes Service (AKS)

On AKS, you install the agent through the Amazon CloudWatch Observability Helm chart. The cluster authenticates to AWS with workload identity that is federated to an AWS IAM role.

Automated setup (recommended)

The onboarding scripts automate the setup. The scripts configure the cluster's workload identity, install the chart, and create the AWS IAM role and trust policy. The chart installs the agent with a default OpenTelemetry configuration. This configuration enables OTel Container Insights and starts an OTLP receiver for metrics, logs, and traces from applications in the cluster. The agent enriches all of this telemetry and forwards it to the CloudWatch OTLP endpoints.

Run the Azure step first, then the AWS trust step. The AWS trust step needs the cluster's OIDC issuer URL, which is available only after the Azure step enables the issuer. Provide the role ARN that the AWS trust step creates or updates (by default, `arn:aws:iam::account-id:role/CloudWatchAgentServerRole`).

> **Reusing an existing IAM role**
>
> Both scripts are safe to run even if the IAM role already exists. The AWS trust step merges the Azure trust into the role's existing trust policy instead of replacing it, so other trust statements remain. It attaches `CloudWatchAgentServerPolicy` only if the role doesn't already have it, and leaves the role's other policies unchanged.

To set up the agent on AKS using the onboarding scripts

1. On a machine with the Azure CLI signed in (for example, Azure Cloud Shell), run the Azure step with the role ARN and the AWS Region. The script enables the OIDC issuer and workload identity on the cluster, installs the agent, and prints the cluster's OIDC issuer URL.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/azure/setup.sh | \
  CWAGENT_PLATFORM=azure_aks \
  CWAGENT_AZURE_RESOURCE_ID=cluster-resource-id \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

2. On a machine with AWS credentials that have IAM write access to the target account (for example, AWS CloudShell), run the AWS trust step with the OIDC issuer URL from the previous step. The script creates the role, attaches `CloudWatchAgentServerPolicy`, and federates the cluster issuer.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/aws/setup.sh | \
  CWAGENT_PLATFORM=azure_aks \
  CWAGENT_AZURE_OIDC_ISSUER=issuer-url \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

Manual setup

To enable workload identity on the cluster

> **Active Azure CLI subscription**
>
> The following `az` commands act on the subscription that is active in the Azure CLI. Confirm it with `az account show`, or switch with `az account set --subscription` *`subscription-id`*, before you run them.

1. Enable the OIDC issuer and workload identity on the AKS cluster.

```
az aks update \
  --resource-group resource-group \
  --name cluster-name \
  --enable-oidc-issuer \
  --enable-workload-identity
```

2. Retrieve the cluster's OIDC issuer URL to use in the next procedure.

```
az aks show \
  --resource-group resource-group \
  --name cluster-name \
  --query oidcIssuerProfile.issuerUrl --output tsv
```

To create the IAM role in your AWS account

1. Register the cluster's OIDC issuer as an IAM OIDC identity provider. The audience (`--client-id-list`) is `sts.amazonaws.com`.

```
aws iam create-open-id-connect-provider --url issuer-url --client-id-list
 sts.amazonaws.com
```

2. Create a trust policy that allows the agent's service account to assume the role. Save it to a file named `trust-policy.json`. Replace *account-id* with your AWS account ID and *issuer-host* with the issuer URL without the `https://` prefix.

> **Reusing an existing IAM role**
>
> If you reuse an existing role, merge this policy with the role's existing trust policy. Each statement is identified by its cluster-specific OIDC provider principal and its `Condition` block, not by a `Sid`, so you don't need to set or preserve one.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": { "Federated": "arn:aws:iam::account-id:oidc-provider/issuer-
host" },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": { "StringEquals": { "issuer-host:sub":
 "system:serviceaccount:amazon-cloudwatch:cloudwatch-agent", "issuer-host:aud":
 "sts.amazonaws.com" } }
    }
  ]
}
```

3. Create the role and attach the CloudWatchAgentServerPolicy managed policy. Note the role ARN that the first command returns.

```
aws iam create-role --role-name role-name --assume-role-policy-document file://
trust-policy.json
aws iam attach-role-policy --role-name role-name --policy-arn
 arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

To install the agent

1. Add the Amazon CloudWatch Observability Helm chart repository and install the chart. Set `k8sMode`, `roleArn`, `region`, and `clusterName`. Enable OTel Container Insights and assign the default OpenTelemetry configuration to the node agent.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update

helm upgrade --install amazon-cloudwatch-observability aws-observability/amazon-
cloudwatch-observability \
  --set k8sMode=AKS \
  --set roleArn=role-arn \
  --set region=region \
  --set clusterName=cluster-name \
  --set containerInsights.enabled=false \
  --set containerLogs.enabled=false \
  --set otelContainerInsights.enabled=true \
  --set otelContainerInsights.logs.enabled=true \
  --set-string 'agents[0].name=cloudwatch-agent' \
  --set-string 'agents[0].config=default:otel' \
  --set-string 'agents[1].name=cloudwatch-agent-cluster-scraper' \
  --set-string 'agents[1].mode=deployment' \
```

```
  --set-string 'agents[1].config=default' \
  --namespace amazon-cloudwatch --create-namespace
```

2. List every `agents[]` entry as shown. The Helm `--set` flag replaces a whole list element, so if you omit the cluster-scraper entry, you remove it.

Install the CloudWatch agent on Google Cloud

You can run the Amazon CloudWatch agent on Google Cloud to collect metrics, logs, and traces. The agent sends this telemetry to Amazon CloudWatch in your AWS account. This page covers two environments:

- Google Compute Engine (GCE) – You install the agent package directly on the instance.
- Google Kubernetes Engine (GKE) – You install the agent through the Amazon CloudWatch Observability Helm chart.

For each environment, you can either use the onboarding scripts or follow the manual steps.

> **Identifying your instance or cluster**
>
> The onboarding scripts identify your instance or cluster by project, location, and name. The project comes from `CWAGENT_GCP_PROJECT`, or from the Google Cloud CLI default when you omit it. The location (`CWAGENT_GCP_LOCATION`) is the zone for an instance, and the zone or region for a cluster.

How the agent authenticates from Google Cloud

We recommend that you federate a Google Cloud identity to an AWS IAM role so that the agent uses temporary credentials. This approach avoids storing long-lived AWS access keys on the instance or cluster, which reduces your security risk. The exact mechanism depends on the environment:

- On an instance, the attached service account issues OpenID Connect (OIDC) tokens. The agent obtains a token from the GCE metadata server and calls `AssumeRoleWithWebIdentity` to obtain temporary AWS credentials.
- On GKE, the cluster issues the OIDC token through a projected service account token, and the agent assumes the role in the same way.

In both cases, your AWS account needs an IAM role that trusts the Google Cloud identity and has the CloudWatchAgentServerPolicy managed policy attached.

> **Google is a built-in identity provider**
>
> For instances, you don't create an IAM OIDC identity provider. IAM has Google built in, so the trust policy names `accounts.google.com` directly. For GKE, you do register the cluster's OIDC issuer as an IAM OIDC identity provider.

Google Compute Engine (GCE)

On a GCE instance, you install the agent package in the same way as on an on-premises server. The instance authenticates to AWS with its attached service account, which is federated to an AWS IAM role.

> **The instance needs a service account**
>
> Instances receive the project's default service account when you create them, unless you opt out. Attaching a service account to an existing instance requires stopping the instance first, so neither the scripts nor the manual steps change it for you.

Automated setup (recommended)

The CloudWatch agent provides onboarding scripts that automate the setup. The scripts read the instance's service account, install the agent, and create the AWS IAM role and trust policy. The scripts configure the agent with a default OpenTelemetry configuration. This configuration collects host metrics (such as CPU, memory, disk, and network). It also starts an OpenTelemetry Protocol (OTLP) receiver for metrics, logs, and traces from applications on the instance. The agent enriches all of this telemetry and forwards it to the CloudWatch OTLP endpoints.

Run the Google Cloud step first, then the AWS trust step. Provide the role ARN that the AWS trust step creates or updates (by default, `arn:aws:iam::`*`account-id`*`:role/CloudWatchAgentServerRole`).

> **Reusing an existing IAM role**
>
> Both scripts are safe to run even if the IAM role already exists. The AWS trust step merges the Google trust into the role's existing trust policy instead of replacing it, so other trust

statements remain. It attaches `CloudWatchAgentServerPolicy` only if the role doesn't already have it, and leaves the role's other policies unchanged.

To set up the agent on a GCE instance using the onboarding scripts

1. On a machine with the Google Cloud CLI signed in (for example, Cloud Shell), run the Google Cloud step with the role ARN and the AWS Region. The script reads the instance's service account, installs and starts the agent, and prints the service account unique ID.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/gcp/setup.sh | \
  CWAGENT_PLATFORM=gcp_gce \
  CWAGENT_GCP_LOCATION=zone \
  CWAGENT_GCP_INSTANCE_NAME=instance-name \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

2. On a machine with AWS credentials that have IAM write access to the target account (for example, AWS CloudShell), run the AWS trust step with the service account unique ID from the previous step. The script creates the role, attaches `CloudWatchAgentServerPolicy`, and adds the Google web-identity trust.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/aws/setup.sh | \
  CWAGENT_PLATFORM=gcp_gce \
  CWAGENT_GCP_SA_UNIQUE_ID=unique-id \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

Manual setup

To find the instance's service account

Active Google Cloud project

The following `gcloud compute` commands act on the project that is active in the Google Cloud CLI. Confirm it with `gcloud config get-value project`, or add `--project`

project-id, before you run them. Don't scope `gcloud iam service-accounts describe` to a project. A service account email is globally unique, and the account can belong to a different project than the instance.

1. Retrieve the email of the service account attached to the instance. If the command returns nothing, the instance has no service account.

```
gcloud compute instances describe instance-name --zone zone --format
 'value(serviceAccounts[0].email)'
```

2. Retrieve the service account's unique ID to use in the next procedure. The trust policy matches on this value, not on the email address.

```
gcloud iam service-accounts describe service-account-email --format
 'value(uniqueId)'
```

To create the IAM role in your AWS account

1. Create a trust policy that allows the service account to assume the role. Save it to a file named `trust-policy.json`. Replace *unique-id* with the service account unique ID.

Reusing an existing IAM role (GCE)

If you reuse an existing role, merge this policy with the role's existing trust policy. Keep the `Sid` value. Every GCE statement uses the same `accounts.google.com` principal, so the `Sid` is what identifies the statement for this service account. A statement without one is replaced the next time you run the onboarding scripts.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GCEunique-id",
      "Effect": "Allow",
      "Principal": { "Federated": "accounts.google.com" },
      "Action": "sts:AssumeRoleWithWebIdentity",
```

```
      "Condition": { "StringEquals": { "accounts.google.com:aud": "unique-
id", "accounts.google.com:sub": "unique-id", "accounts.google.com:oaud":
 "sts.amazonaws.com" } }
    }
  ]
}
```

To onboard more than one service account to the same role, add a separate statement for each one, with its own `Sid` and unique ID.

On tokens that Google issues to a service account, IAM matches `sub` against the service account unique ID. It matches `aud` against the token's authorized party (`azp`), which Google also sets to the unique ID. It matches `oaud` against the audience that the agent requested the token for, which is `sts.amazonaws.com`.

2. Create the role and attach the CloudWatchAgentServerPolicy managed policy. Note the role ARN that the first command returns.

```
aws iam create-role --role-name role-name --assume-role-policy-document file://
trust-policy.json
aws iam attach-role-policy --role-name role-name --policy-arn
 arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

To install and start the agent

1. Download and install the agent on the instance, in the same way as on an on-premises server. See Install the CloudWatch agent on on-premises servers.
2. Set the AWS Region and the ARN of the role to assume, then start the agent with the default OpenTelemetry configuration (`default:otel`). The default OpenTelemetry configuration reads the role ARN from the `CWAGENT_ROLE_ARN` environment variable.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a set-env -e
 AWS_REGION=region
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a set-env -e
 CWAGENT_ROLE_ARN=role-arn
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m auto -c default:otel -s
```

Google Kubernetes Engine (GKE)

On GKE, you install the agent through the Amazon CloudWatch Observability Helm chart. The cluster authenticates to AWS with a projected service account token that is federated to an AWS IAM role.

Automated setup (recommended)

The onboarding scripts automate the setup. The scripts read the cluster, install the chart, and create the AWS IAM role and trust policy. The chart installs the agent with a default OpenTelemetry configuration. This configuration enables OTel Container Insights. It also starts an OTLP receiver for metrics, logs, and traces from applications in the cluster. The agent enriches all of this telemetry and forwards it to the CloudWatch OTLP endpoints.

Run the Google Cloud step first, then the AWS trust step. The AWS trust step needs the cluster's OIDC issuer URL, which the Google Cloud step prints. Provide the role ARN that the AWS trust step creates or updates (by default, `arn:aws:iam::`*`account-id`*`:role/CloudWatchAgentServerRole`).

> **Reusing an existing IAM role**
>
> Both scripts are safe to run even if the IAM role already exists. The AWS trust step merges the Google trust into the role's existing trust policy instead of replacing it, so other trust statements remain. It attaches `CloudWatchAgentServerPolicy` only if the role doesn't already have it, and leaves the role's other policies unchanged.

To set up the agent on GKE using the onboarding scripts

1. On a machine with the Google Cloud CLI signed in (for example, Cloud Shell), run the Google Cloud step with the role ARN and the AWS Region. The script installs the agent and prints the cluster's OIDC issuer URL.

   > **Tools the script needs to install the chart**
   >
   > The script installs the chart when `helm`, `kubectl`, and `gke-gcloud-auth-plugin` are all available. Otherwise it prints the commands for you to run from a shell that has them.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/gcp/setup.sh | \
  CWAGENT_PLATFORM=gcp_gke \
  CWAGENT_GCP_LOCATION=location \
  CWAGENT_K8S_CLUSTER_NAME=cluster-name \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

2. On a machine with AWS credentials that have IAM write access to the target account (for example, AWS CloudShell), run the AWS trust step with the OIDC issuer URL from the previous step. The script creates the role, attaches `CloudWatchAgentServerPolicy`, and federates the cluster issuer.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/aws/setup.sh | \
  CWAGENT_PLATFORM=gcp_gke \
  CWAGENT_GCP_OIDC_ISSUER=issuer-url \
  CWAGENT_AWS_ROLE_ARN=role-arn \
  CWAGENT_AWS_REGION=region \
  sh
```

Manual setup

To find the cluster's OIDC issuer URL

> **Active Google Cloud project**
>
> The following `gcloud` commands act on the project that is active in the Google Cloud CLI. Confirm it with `gcloud config get-value project`, or add `--project` *`project-id`* to each command, before you run them.

> **GKE serves the issuer by default**
>
> Every GKE cluster serves its OIDC discovery document, so there is no setting to enable. You construct the issuer URL from the project, location, and cluster name.

1. Confirm that you can read the cluster, and note its project, location, and name.

```
gcloud container clusters describe cluster-name --location location --format
 'value(name)'
```

2. Build the issuer URL in the following form. Use `locations` for both zonal and regional clusters.

```
https://container.googleapis.com/v1/projects/project-id/locations/location/
clusters/cluster-name
```

To create the IAM role in your AWS account

1. Register the cluster's OIDC issuer as an IAM OIDC identity provider. The audience (`--client-id-list`) is `sts.amazonaws.com`.

```
aws iam create-open-id-connect-provider --url issuer-url --client-id-list
 sts.amazonaws.com
```

2. Create a trust policy that allows the agent's service account to assume the role. Save it to a file named `trust-policy.json`. Replace *account-id* with your AWS account ID and *issuer-host* with the issuer URL without the `https://` prefix.

> **Reusing an existing IAM role (GKE)**
>
> If you reuse an existing role, merge this policy with the role's existing trust policy. Each statement is identified by its cluster-specific OIDC provider principal and its `Condition` block, not by a `Sid`, so you don't need to set or preserve one.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Federated": "arn:aws:iam::account-id:oidc-provider/issuer-
host" },
      "Action": "sts:AssumeRoleWithWebIdentity",
```

```
      "Condition": { "StringEquals": { "issuer-host:sub":
 "system:serviceaccount:amazon-cloudwatch:cloudwatch-agent", "issuer-host:aud":
 "sts.amazonaws.com" } }
    }
  ]
}
```

3. Create the role and attach the CloudWatchAgentServerPolicy managed policy. Note the role ARN that the first command returns.

```
aws iam create-role --role-name role-name --assume-role-policy-document file://
trust-policy.json
aws iam attach-role-policy --role-name role-name --policy-arn
 arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

To install the agent

1. Configure `kubectl` to use the cluster. This command requires `gke-gcloud-auth-plugin`.

```
gcloud container clusters get-credentials cluster-name --location location
```

2. Add the Amazon CloudWatch Observability Helm chart repository and install the chart. Set `k8sMode`, `roleArn`, `region`, and `clusterName`. Enable OTel Container Insights and assign the default OpenTelemetry configuration to the node agent.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update

helm upgrade --install amazon-cloudwatch-observability aws-observability/amazon-
cloudwatch-observability \
  --set k8sMode=GKE \
  --set roleArn=role-arn \
  --set region=region \
  --set clusterName=cluster-name \
  --set containerInsights.enabled=false \
  --set containerLogs.enabled=false \
  --set otelContainerInsights.enabled=true \
  --set otelContainerInsights.logs.enabled=true \
  --set-string 'agents[0].name=cloudwatch-agent' \
  --set-string 'agents[0].config=default:otel' \
  --set-string 'agents[1].name=cloudwatch-agent-cluster-scraper' \
```

```
  --set-string 'agents[1].mode=deployment' \
  --set-string 'agents[1].config=default' \
  --namespace amazon-cloudwatch --create-namespace
```

3. List every `agents[]` entry as shown. The Helm `--set` flag replaces a whole list element, so if you omit the cluster-scraper entry, you remove it.

Install the CloudWatch agent on new instances using CloudFormation

This section describes how to install the CloudWatch agent on new Amazon EC2 instances using AWS CloudFormation.

> **Note**
>
> Amazon uploaded several CloudFormation templates to GitHub that can help you install and update the CloudWatch agent on new Amazon EC2 instances. For more information about using CloudFormation, see What is AWS CloudFormation?.

The template location is Deploy the Amazon CloudWatch agent to EC2 instances using CloudFormation. This location includes both `inline` and `ssm` directories. Each of these directories contains templates for both Linux and Windows instances.

- The templates in the `inline` directory have the CloudWatch agent configuration embedded into the CloudFormation template. By default, the Linux templates collect the metrics `mem_used_percent` and `swap_used_percent`, and the Windows templates collect `Memory % Committed Bytes In Use` and `Paging File % Usage`.

  To modify these templates to collect different metrics, modify the following section of the template. The following example is from the template for Linux servers. Follow the format and syntax of the agent configuration file to make these changes. For more information, see Manually create or edit the CloudWatch agent configuration file.

```
{
   "metrics":{
      "append_dimensions":{
         "AutoScalingGroupName":"${!aws:AutoScalingGroupName}",
         "ImageId":"${!aws:ImageId}",
         "InstanceId":"${!aws:InstanceId}",
```

```
            "InstanceType":"${!aws:InstanceType}"
         },
         "metrics_collected":{
            "mem":{
               "measurement":[
                  "mem_used_percent"
               ]
            },
            "swap":{
               "measurement":[
                  "swap_used_percent"
               ]
            }
         }
      }
   }
```

> **Note**
>
> In the inline templates, all placeholder variables must have an exclamation mark (!) before them as an escape character. You can see this in the example template. If you add other placeholder variables, be sure to add an exclamation mark before the name.

- The templates in the `ssm` directory load an agent configuration file from Parameter Store. To use these templates, you must first create a configuration file and upload it to Parameter Store. You then provide the Parameter Store name of the file in the template. You can create the configuration file manually or by using the wizard. For more information, see Create the CloudWatch agent configuration file.

You can use both types of templates for installing the CloudWatch agent and for updating the agent configuration.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

Tutorial: Install and configure the CloudWatch agent using an CloudFormation inline template

This tutorial walks you through using CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using the inline templates, which don't require the use of the JSON configuration file or Parameter Store.

The inline template includes the agent configuration in the template. In this tutorial, you use the default agent configuration contained in the template.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use CloudFormation to install the CloudWatch agent on a new instance

1. Download the template from GitHub. In this tutorial, download the inline template for Amazon Linux 2 as follows:

   ```
   curl -O https://raw.githubusercontent.com/aws-cloudformation/aws-cloudformation-templates/main/Solutions/AmazonCloudWatchAgent/inline/amazon_linux.yaml
   ```

2. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. Choose **Create stack**.
4. For **Choose a template**, select **Upload a template to Amazon S3**, choose the downloaded template, and choose **Next**.
5. On the **Specify Details** page, fill out the following parameters and choose **Next**:

   - **Stack name**: Choose a stack name for your CloudFormation stack.
   - **IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics, logs, and traces. For more information, see Prerequisites.
   - **InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
   - **InstanceType**: Choose a valid instance type.
   - **KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.
   - **SSHLocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.

6. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

   If you refresh the console, you see that the new stack has the `CREATE_IN_PROGRESS` status.

8. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

   Use the following command to confirm that the agent is installed:

   ```
   rpm -qa amazon-cloudwatch-agent
   ```

   Use the following command to confirm that the agent is running:

   ```
   ps aux | grep amazon-cloudwatch-agent
   ```

The next procedure demonstrates using CloudFormation to update the CloudWatch agent using an inline template. The default inline template collects the `mem_used_percent` metric. In this tutorial, you change the agent configuration to stop collecting that metric.

To use CloudFormation to update the CloudWatch agent

1. In the template that you downloaded in the previous procedure, remove the following lines and then save the template:

   ```
   "mem": {

        "measurement": [
            "mem_used_percent"
          ]
    },
   ```

2. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. On the CloudFormation dashboard, select the stack that you created and choose **Update Stack**.
4. For **Select Template**, select **Upload a template to Amazon S3**, choose the template that you modified, and choose **Next**.
5. On the **Options** page, choose **Next** and then **Next**.
6. On the **Review** page, review your information and choose **Update**.

   After some time, you see `UPDATE_COMPLETE`.

Tutorial: Install the CloudWatch agent using CloudFormation and Parameter Store

This tutorial walks you through using CloudFormation to install the CloudWatch agent on a new Amazon EC2 instance. This tutorial installs on a new instance running Amazon Linux 2 using an agent configuration file that you create and save in Parameter Store.

After the procedure for installing the agent, the tutorial continues with how to update the agent.

To use CloudFormation to install the CloudWatch agent on a new instance using a configuration from Parameter Store

1. If you haven't done so already, download the CloudWatch agent package to one of your computers so that you can create the agent configuration file. For more information and downloading the agent using Parameter Store, see Download the CloudWatch agent package.
2. Create the agent configuration file and save it in Parameter Store. For more information, see Create the CloudWatch agent configuration file.
3. Download the template from GitHub as follows:

   ```
   curl -O https://raw.githubusercontent.com/awslabs/aws-cloudformation-templates/
   master/aws/solutions/AmazonCloudWatchAgent/ssm/amazon_linux.template
   ```

4. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
5. Choose **Create stack**.
6. For **Choose a template**, select **Upload a template to Amazon S3**, choose the template that you downloaded, and choose **Next**.
7. On the **Specify Details** page, fill out the following parameters accordingly and choose **Next**:

   - **Stack name**: Choose a stack name for your CloudFormation stack.
   - **IAMRole**: Choose an IAM role that has permissions to write CloudWatch metrics, logs, and traces. For more information, see Prerequisites.
   - **InstanceAMI**: Choose an AMI that is valid in the Region where you're going to launch your stack.
   - **InstanceType**: Choose a valid instance type.
   - **KeyName**: To enable SSH access to the new instance, choose an existing Amazon EC2 key pair. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.

- **SSHLocation**: Specifies the IP address range that can be used to connect to the instance using SSH. The default allows access from any IP address.
- **SSMKey**: Specifies the agent configuration file that you created and saved in Parameter Store.

8. On the **Options** page, you can choose to tag your stack resources. Choose **Next**.
9. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create**.

   If you refresh the console, you see that the new stack has the `CREATE_IN_PROGRESS` status.
10. When the instance is created, you can see it in the Amazon EC2 console. Optionally, you can connect to the host and check the progress.

    Use the following command to confirm that the agent is installed:

    ```
    rpm -qa amazon-cloudwatch-agent
    ```

    Use the following command to confirm that the agent is running:

    ```
    ps aux | grep amazon-cloudwatch-agent
    ```

The next procedure demonstrates using CloudFormation to update the CloudWatch agent, using an agent configuration that you saved in Parameter Store.

To use CloudFormation to update the CloudWatch agent using a configuration in Parameter Store

1. Change the agent configuration file stored in Parameter Store to the new configuration that you want.
2. In the CloudFormation template that you downloaded in the the section called “Tutorial: Install the CloudWatch agent using CloudFormation and Parameter Store” topic, change the version number. For example, you might change `VERSION=1.0` to `VERSION=2.0`.
3. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
4. On the CloudFormation dashboard, select the stack that you created and choose **Update Stack**.

5. For **Select Template**, select **Upload a template to Amazon S3**, select the template that you just modified, and choose **Next**.
6. On the **Options** page, choose **Next** and then **Next**.
7. On the **Review** page, review your information and choose **Update**.

   After some time, you see `UPDATE_COMPLETE`.

Troubleshooting installation of the CloudWatch agent with CloudFormation

This section helps you troubleshoot issues with installing and updating the CloudWatch agent using CloudFormation.

Detecting when an update fails

If you use CloudFormation to update your CloudWatch agent configuration, and use an invalid configuration, the agent stops sending any metrics to CloudWatch. A quick way to check whether an agent configuration update succeeded is to look at the `cfn-init-cmd.log` file. On a Linux server, the file is located at `/var/log/cfn-init-cmd.log`. On a Windows instance, the file is located at `C:\cfn\log\cfn-init-cmd.log`.

Metrics are missing

If you don't see metrics that you expect to see after installing or updating the agent, confirm that the agent is configured to collect that metric. To do this, check the `amazon-cloudwatch-agent.json` file to make sure that the metric is listed, and check that you are looking in the correct metric namespace. For more information, see CloudWatch agent files and locations.

Set up the CloudWatch agent with security-enhanced Linux (SELinux)

If your system has security-enhanced Linux (SELinux) enabled, you must apply the appropriate security policies to ensure that the CloudWatch agent runs in a confined domain.

Prerequisites

Before you can configure SELinux for the agent, check the following **prerequisites**:

To complete the prerequisites for using the CloudWatch agent with SELinux

1. If you haven't done so, install the following SELinux policy development packages:

   ```
   sudo yum update
   ```

```
sudo yum install -y selinux-policy-devel policycoreutils-devel rpm-build git
```

2. Run the following command to check your system's SELinux status:

```
sestatus
```

Example output:

```
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   permissive
Mode from config file:          permissive
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      33
```

If you find that SELinux is currently disabled, do the following:

a. Open the SELinux file by entering the following command:

```
sudo vi /etc/selinux/config
```

b. Set the `SELINUX` parameter to either `permissive` or `enforcing`. For example:

```
SELINUX=enforcing
```

c. Save the file and reboot the system to apply the changes.

```
sudo reboot
```

3. Ensure that the CloudWatch agent is running as a `systemd` service. This is required to use it within a confined SELinux domain.

```
sudo systemctl status amazon-cloudwatch-agent
```

If the agent is correctly configured, the output should indicate that it is `active (running)` and `enabled` at startup.

Configure SELinux for the agent

After you complete the prerequisites, you can configure SELinux.

To configure SELinux for the CloudWatch agent

1. Clone the SELinux policy for the CloudWatch agent by entering the following command:

```
git clone https://github.com/aws/amazon-cloudwatch-agent-selinux.git
```

2. Navigate to the cloned repository and then update the script permissions by entering the following commands:

```
cd amazon-cloudwatch-agent-selinux
chmod +x amazon_cloudwatch_agent.sh
```

3. Use sudo to run the SELinux policy installation script by entering the following command. During execution, the script prompts you to enter y or n to allow automatic restart. This restart ensures that the agent transitions into the correct SELinux domain.

```
sudo ./amazon_cloudwatch_agent.sh
```

4. If the CloudWatch agent hasn't been restarted yet, restart it to ensure that it transitions to the correct SELinux domain:

```
sudo systemctl restart amazon-cloudwatch-agent
```

5. Verify that CloudWatch Agent is running in the confined domain by entering the following command:

```
ps -efZ | grep amazon-cloudwatch-agent
```

   If the agent is correctly confined, the output should indicate a SELinux-confined domain instead of `unconfined_service_t`.

   The following is an example of output when the agent is correctly confined.

```
system_u:system_r:confined_t:s0 root 1234 1 0 12:00 ? 00:00:10 /opt/aws/amazon-
cloudwatch-agent/bin/amazon-cloudwatch-agent
```

After SELinux is configured, you can proceed to configure the agent to collect metrics, logs, and traces. For more information, see Manually create or edit the CloudWatch agent configuration file.

Create the CloudWatch agent configuration file

Before running the CloudWatch agent on any servers, you must create one or more CloudWatch agent configuration files.

The agent configuration file is a JSON file that specifies the metrics, logs, and traces that the agent is to collect, including custom metrics. You can create it by using the wizard or by creating it yourself from scratch. You could also use the wizard to initially create the configuration file and then modify it manually. If you create or modify the file manually, the process is more complex, but you have more control over the metrics collected and can specify metrics not available through the wizard.

Any time you change the agent configuration file, you must then restart the agent to have the changes take effect. To restart the agent, follow the instructions in (Optional) Modify the common configuration and named profile for CloudWatch agent.

After you have created a configuration file, you can save it manually as a JSON file and then use this file when installing the agent on your servers. Alternatively, you can store it in Systems Manager Parameter Store if you're going to use Systems Manager when you install the agent on servers.

The CloudWatch agent supports using multiple configuration files. For more information, see Creating multiple CloudWatch agent configuration files.

Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see Amazon CloudWatch Pricing.

Contents

- Create the CloudWatch agent configuration file with the wizard
- Creating multiple CloudWatch agent configuration files
- Manually create or edit the CloudWatch agent configuration file

Create the CloudWatch agent configuration file with the wizard

The agent configuration file wizard, `amazon-cloudwatch-agent-config-wizard`, asks a series of questions to help you configure the CloudWatch agent for your needs. This section describes

the credentials required for the configuration file. It describes how to run the CloudWatch agent configuration wizard. It also describes the metrics that are predefined in the wizard.

Required credentials

The wizard can autodetect the credentials and AWS Region to use if you have the AWS credentials and configuration files in place before you start the wizard. For more information about these files, see Configuration and Credential Files in the *AWS Systems Manager User Guide*.

In the AWS credentials file, the wizard checks for default credentials and also looks for an `AmazonCloudWatchAgent` section such as the following:

```
[AmazonCloudWatchAgent]
aws_access_key_id = my_access_key
aws_secret_access_key = my_secret_key
```

The wizard displays the default credentials, the credentials from the `AmazonCloudWatchAgent`, and an `Others` option. You can select which credentials to use. If you choose `Others`, you can input credentials.

For *my_access_key* and *my_secret_key*, use the keys from the IAM user that has the permissions to write to Systems Manager Parameter Store.

In the AWS configuration file, you can specify the Region that the agent sends metrics to if it's different than the `[default]` section. The default is to publish the metrics to the Region where the Amazon EC2 instance is located. If the metrics should be published to a different Region, specify the Region here. In the following example, the metrics are published to the `us-west-1` Region.

```
[AmazonCloudWatchAgent]
region = us-west-1
```

Run the CloudWatch agent configuration wizard

To create the CloudWatch agent configuration file

1. Start the CloudWatch agent configuration wizard by entering the following:

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
   ```

On a server running Windows Server, run the following commands to launch the wizard:

```
cd "C:\Program Files\Amazon\AmazonCloudWatchAgent"
```

```
.\amazon-cloudwatch-agent-config-wizard.exe
```

2. Answer the questions to customize the configuration file for your server.
3. If you're storing the configuration file locally, the configuration file `config.json` is stored in `/opt/aws/amazon-cloudwatch-agent/bin/` on Linux servers, and is stored in `C:\Program Files\Amazon\AmazonCloudWatchAgent` on Windows Server. You can then copy this file to other servers where you want to install the agent.

   If you're going to use Systems Manager to install and configure the agent, be sure to answer **Yes** when prompted whether to store the file in Systems Manager Parameter Store. You can also choose to store the file in Parameter Store even if you aren't using the SSM Agent to install the CloudWatch agent. To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions.

CloudWatch agent predefined metric sets

The wizard is configured with predefined sets of metrics, with different detail levels. These sets of metrics are shown in the following tables. For more information about these metrics, see Metrics collected by the CloudWatch agent.

> **Note**
>
> Parameter Store supports parameters in Standard and Advanced tiers. These parameter tiers are not related to the Basic, Standard, and Advanced levels of metric details that are described in these tables.

Amazon EC2 instances running Linux

Detail level	Metrics included
Basic	**Mem:** mem_used_percent

Detail level	Metrics included
	Disk: disk_used_percent The `disk` metrics such as `disk_used_percent` have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.
Standard	**CPU:** `cpu_usage_idle`, `cpu_usage_iowait`, `cpu_usage_user`, `cpu_usage_system` **Disk:** `disk_used_percent`, `disk_inodes_free` **Diskio:** `diskio_io_time` **Mem:** `mem_used_percent` **Swap:** `swap_used_percent`
Advanced	**CPU:** `cpu_usage_idle`, `cpu_usage_iowait`, `cpu_usage_user`, `cpu_usage_system` **Disk:** `disk_used_percent`, `disk_inodes_free` **Diskio:** `diskio_io_time`, `diskio_write_bytes`, `diskio_read_bytes`, `diskio_writes`, `diskio_reads` **Mem:** `mem_used_percent` **Netstat:** `netstat_tcp_established`, `netstat_tcp_time_wait` **Swap:** `swap_used_percent`

On-premises servers running Linux

Detail level	Metrics included
Basic	**Disk:** `disk_used_percent`

Detail level	Metrics included
	Diskio: diskio_write_bytes ,diskio_read_bytes ,diskio_writes ,diskio_reads **Mem:** mem_used_percent **Net:** net_bytes_sent ,net_bytes_recv ,net_packets_sent ,net_packets_recv **Swap:** swap_used_percent
Standard	**CPU:** cpu_usage_idle ,cpu_usage_iowait **Disk:** disk_used_percent ,disk_inodes_free **Diskio:** diskio_io_time ,diskio_write_bytes ,diskio_read_bytes ,diskio_writes ,diskio_reads **Mem:** mem_used_percent **Net:** net_bytes_sent ,net_bytes_recv ,net_packets_sent ,net_packets_recv **Swap:** swap_used_percent

Detail level	Metrics included
Advanced	**CPU:** `cpu_usage_guest` , `cpu_usage_idle` , `cpu_usage_iowait` , `cpu_usage_steal` , `cpu_usage_user` , `cpu_usage_system` **Disk:** `disk_used_percent` , `disk_inodes_free` **Diskio:** `diskio_io_time` , `diskio_write_bytes` , `diskio_read_bytes` , `diskio_writes` , `diskio_reads` **Mem:** `mem_used_percent` **Net:** `net_bytes_sent` , `net_bytes_recv` , `net_packets_sent` , `net_packets_recv` **Netstat:** `netstat_tcp_established` , `netstat_tcp_time_wait` **Swap:** `swap_used_percent`

Amazon EC2 instances running Windows Server

> **Note**
>
> The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for `LogicalDisk % Free Space` is just `% Free Space`.

Detail level	Metrics included
Basic	**Memory:** `Memory % Committed Bytes In Use` **LogicalDisk:** `LogicalDisk % Free Space`
Standard	**Memory:** `Memory % Committed Bytes In Use` **Paging:** `Paging File % Usage`

Detail level	Metrics included
	Processor: `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **PhysicalDisk:** `PhysicalDisk % Disk Time` **LogicalDisk:** `LogicalDisk % Free Space`
Advanced	**Memory:** `Memory % Committed Bytes In Use` **Paging:** `Paging File % Usage` **Processor:** `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **LogicalDisk:** `LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk % Disk Time` , `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **TCP:** `TCPv4 Connections Established` , `TCPv6 Connections Established`

On-premises server running Windows Server

> **Note**
>
> The metric names listed in this table display how the metric appears when viewed in the console. The actual metric name might not include the first word. For example, the actual metric name for `LogicalDisk % Free Space` is just `% Free Space`.

Detail level	Metrics included
Basic	**Paging:** `Paging File % Usage`

Detail level	Metrics included
	Processor: Processor % Processor Time **LogicalDisk:**LogicalDisk % Free Space **PhysicalDisk:** PhysicalDisk Disk Write Bytes/sec ,PhysicalDisk Disk Read Bytes/sec ,PhysicalDisk Disk Writes/sec , PhysicalDisk Disk Reads/sec **Memory:** Memory % Committed Bytes In Use **Network Interface:** Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec ,Network Interface Packets Sent/sec, Network Interface Packets Received/sec
Standard	**Paging:** Paging File % Usage **Processor:** Processor % Processor Time, Processor % Idle Time, Processor % Interrupt Time **LogicalDisk:** LogicalDisk % Free Space **PhysicalDisk:** PhysicalDisk % Disk Time ,PhysicalDisk Disk Write Bytes/sec ,PhysicalDisk Disk Read Bytes/sec , PhysicalDisk Disk Writes/sec ,PhysicalDisk Disk Reads/sec **Memory:** Memory % Committed Bytes In Use **Network Interface:** Network Interface Bytes Sent/sec, Network Interface Bytes Received/sec ,Network Interface Packets Sent/sec, Network Interface Packets Received/sec

Detail level	Metrics included
Advanced	**Paging:**`Paging File % Usage` **Processor:** `Processor % Processor Time`, `Processor % Idle Time`, `Processor % Interrupt Time`, `Processor % User Time` **LogicalDisk:** `LogicalDisk % Free Space` **PhysicalDisk:** `PhysicalDisk % Disk Time` , `PhysicalDisk Disk Write Bytes/sec` , `PhysicalDisk Disk Read Bytes/sec` , `PhysicalDisk Disk Writes/sec` , `PhysicalDisk Disk Reads/sec` **Memory:** `Memory % Committed Bytes In Use` **Network Interface:** `Network Interface Bytes Sent/sec`, `Network Interface Bytes Received/sec` , `Network Interface Packets Sent/sec`, `Network Interface Packets Received/sec` **TCP:** `TCPv4 Connections Established` , `TCPv6 Connections Established`

Examples of configuration files

Basic system metrics configuration

```
{
  "agent": {
    "metrics_collection_interval": 60,
    "region": "us-east-1"
  },
  "metrics": {
    "namespace": "MySystem",
    "metrics_collected": {
      "cpu": {
        "resources": ["*"],
        "measurement": ["usage_active", "usage_system", "usage_user"]
      },
      "mem": {
```

```
        "measurement": ["used_percent"]
      },
      "disk": {
        "resources": ["/"],
        "measurement": ["used_percent"]
      }
    },
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    }
  }
}
```

Web server monitoring configuration

```
{
  "agent": {
    "metrics_collection_interval": 60,
    "region": "us-east-1"
  },
  "metrics": {
    "namespace": "WebServer",
    "metrics_collected": {
      "cpu": {
        "resources": ["*"],
        "measurement": ["usage_active"]
      },
      "mem": {
        "measurement": ["used_percent"]
      },
      "net": {
        "resources": ["eth0"],
        "measurement": ["bytes_sent", "bytes_recv"]
      }
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/var/log/apache2/access.log",
            "log_group_name": "apache-access-logs",
```

```
            "log_stream_name": "{instance_id}-access"
          },
          {
            "file_path": "/var/log/apache2/error.log",
            "log_group_name": "apache-error-logs",
            "log_stream_name": "{instance_id}-error"
          }
        ]
      }
    }
  }
}
```

Database server configuration

```
{
  "agent": {
    "metrics_collection_interval": 30,
    "region": "us-east-1"
  },
  "metrics": {
    "namespace": "DatabaseServer",
    "metrics_collected": {
      "cpu": {
        "resources": ["*"],
        "measurement": ["usage_active", "usage_iowait"]
      },
      "mem": {
        "measurement": ["used_percent", "available_percent"]
      },
      "disk": {
        "resources": ["/", "/data"],
        "measurement": ["used_percent", "inodes_free"]
      },
      "diskio": {
        "resources": ["*"],
        "measurement": ["read_bytes", "write_bytes", "io_time"]
      }
    },
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
```

```
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    }
  }
}
```

Creating multiple CloudWatch agent configuration files

On both Linux servers and Windows servers, you can set up the CloudWatch agent to use multiple configuration files. For example, you can use a common configuration file that collects a set of metrics, logs, and traces that you always want to collect from all servers in your infrastructure. You can then use additional configuration files that collect metrics from certain applications or in certain situations.

To set this up, first create the configuration files that you want to use. Any configuration files that will be used together on the same server must have different file names. You can store the configuration files on servers or in Parameter Store.

Start the CloudWatch agent using the `fetch-config` option and specify the first configuration file. To append the second configuration file to the running agent, use the same command but with the `append-config` option. All metrics, logs, and traces listed in either configuration file are collected. The following example commands illustrate this scenario using configurations stores as files. The first line starts the agent using the `infrastructure.json` configuration file, and the second line appends the `app.json` configuration file.

The following example commands are for Linux.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2
 -s -c file:/tmp/infrastructure.json
```

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -m
 ec2 -s -c file:/tmp/app.json
```

The following example commands are for Windows Server.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a fetch-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\infrastructure.json"
```

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
 -a append-config -m ec2 -s -c file:"C:\Program Files\Amazon\AmazonCloudWatchAgent
\app.json"
```

The following example configuration files illustrate a use for this feature. The first configuration file is used for all servers in the infrastructure, and the second collects only logs from a certain application and is appended to servers running that application.

infrastructure.json

```
{
  "metrics": {
    "metrics_collected": {
      "cpu": {
        "resources": [
          "*"
        ],
        "measurement": [
          "usage_active"
        ],
        "totalcpu": true
      },
      "mem": {
         "measurement": [
           "used_percent"
        ]
      }
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log"
          },
          {
            "file_path": "/var/log/messages",
            "log_group_name": "/var/log/messages"
          }
        ]
```

```
      }
    }
  }
}
```

app.json

```
{
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/app/app.log*",
                        "log_group_name": "/app/app.log"
                    }
                ]
            }
        }
    }
}
```

Any configuration files appended to the configuration must have different file names from each other and from the initial configuration file. If you use `append-config` with a configuration file with the same file name as a configuration file that the agent is already using, the append command overwrites the information from the first configuration file instead of appending to it. This is true even if the two configuration files with the same file name are on different file paths.

The preceding example shows the use of two configuration files, but there is no limit to the number of configuration files that you can append to the agent configuration. You can also mix the use of configuration files located on servers and configurations located in Parameter Store.

Manually create or edit the CloudWatch agent configuration file

The CloudWatch agent configuration file is a JSON file with five sections: `agent`, `metrics`, `logs`, `traces`, and `opentelemetry`.

- The `agent` section includes fields for the overall configuration of the agent.
- The `metrics` section specifies the custom metrics for collection and publishing to CloudWatch. If you're using the agent only to collect logs, you can omit the `metrics` section from the file.

- The `logs` section specifies what log files are published to CloudWatch Logs. This can include events from the Windows Event Log if the server runs Windows Server.
- The `traces` section specifies the sources for traces that are collected and sent to AWS X-Ray.
- The `opentelemetry` section specifies the sources of metrics, logs, and traces for the agent to collect and send to the CloudWatch OpenTelemetry Protocol (OTLP) endpoints. For more information, see OTLP Endpoints.

This section explains the structure and fields of the CloudWatch agent configuration file. You can view the schema definition for this configuration file. The schema definition is located at *`installation-directory`*`/doc/amazon-cloudwatch-agent-schema.json` on Linux servers and at *`installation-directory`*`/amazon-cloudwatch-agent-schema.json` on servers running Windows Server.

If you create or edit the agent configuration file manually, you can give it any name. For simplicity in troubleshooting, we recommend that you name it `/opt/aws/amazon-cloudwatch-agent/etc/cloudwatch-agent.json` on a Linux server and `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json` on servers running Windows Server. After you have created the file, you can copy it to other servers where you want to install the agent.

When the agent is started, it creates a copy of each configuration file in `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d` directory, with the filename prefixed with either `file_` (for local file sources) or `ssm_` (for Systems Manager parameter store sources) to indicate the configuration origin.

> **Note**
>
> Metrics, logs, and traces collected by the CloudWatch agent incur charges. For more information about pricing, see Amazon CloudWatch Pricing.

CloudWatch agent configuration file: Agent section

The `agent` section can include the following fields. The wizard doesn't create an `agent` section. Instead, the wizard omits it and uses the default values for all fields in this section.

- `metrics_collection_interval` – Optional. Specifies how often all metrics specified in this configuration file are to be collected. You can override this value for specific types of metrics.

This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

The default value is 60.

- `region` – Specifies the Region to use for the CloudWatch endpoint when an Amazon EC2 instance is being monitored. The metrics collected are sent to this Region, such as `us-west-1`. If you omit this field, the agent sends metrics to the Region where the Amazon EC2 instance is located.

  If you are monitoring an on-premises server, this field isn't used, and the agent reads the Region from the `AmazonCloudWatchAgent` profile of the AWS configuration file.

- `credentials` – Specifies an IAM role to use when sending metrics, logs, and traces to a different AWS account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the Amazon Resource Name (ARN) of an IAM role to use for authentication when sending metrics, logs, and traces to a different AWS account. For more information, see Sending metrics, logs, and traces to a different account.
- debug – Optional. Specifies running the CloudWatch agent with debug log messages. The default value is `false`.
- `aws_sdk_log_level` – Optional. Supported only in versions 1.247350.0 and later of the CloudWatch agent.

  You can specify this field to have the agent perform logging for AWS SDK endpoints. The value for this field can include one or more of the following options. Separate multiple options with the `|` character.
  - `LogDebug`
  - `LogDebugWithSigning`
  - `LogDebugWithHTTPBody`
  - `LogDebugRequestRetries`
  - `LogDebugWithEventStreamBody`

  For more information about these options, see LogLevelType.

- `logfile` – Specifies the location where the CloudWatch agent writes log messages. If you specify an empty string, the log goes to stderr. If you don't specify this option, the default locations are the following:
  - Linux: `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log`
  - Windows Server: `c:\\ProgramData\\Amazon\\CloudWatchAgent\\Logs\\amazon-cloudwatch-agent.log`

  The CloudWatch agent automatically rotates the log file that it creates. A log file is rotated out when it reaches 100 MB in size. The agent keeps the rotated log files for up to seven days, and it keeps as many as five backup log files that have been rotated out. Backup log files have a timestamp appended to their filename. The timestamp shows the date and time that the file was rotated out: for example, `amazon-cloudwatch-agent-2018-06-08T21-01-50.247.log.gz`.
- `omit_hostname` – Optional. By default, the hostname is published as a dimension of metrics that are collected by the agent, unless you are using the `append_dimensions` field in the `metrics` section. Set `omit_hostname` to `true` to prevent the hostname from being published as a dimension even if you are not using `append_dimensions`. The default value is `false`.
- `run_as_user` – Optional. Specifies a user to use to run the CloudWatch agent. If you don't specify this parameter, the root user is used. This option is valid only on Linux servers.

  If you specify this option, the user must exist before you start the CloudWatch agent. For more information, see Running the CloudWatch agent as a different user.
- `user_agent` – Optional. Specifies the `user-agent` string that is used by the CloudWatch agent when it makes API calls to the CloudWatch backend. The default value is a string consisting of the agent version, the version of the Go programming language that was used to compile the agent, the runtime operating system and architecture, the build time, and the plugins enabled.
- `usage_data` – Optional. By default, the CloudWatch agent sends health and performance data about itself to CloudWatch whenever it publishes metrics or logs to CloudWatch. This data incurs no costs to you. You can prevent the agent from sending this data by specifying `false` for `usage_data`. If you omit this parameter, the default of `true` is used and the agent sends the health and performance data.

  If you set this value to `false`, you must stop and restart the agent for it to take effect.
- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.

- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.
- `use_dualstack_endpoint` – Optional. If this is `true`, the CloudWatch agent will use dual stack endpoints for all API calls.

The following is an example of an `agent` section.

```
"agent": {
   "metrics_collection_interval": 60,
   "region": "us-west-1",
   "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
   "debug": false,
   "run_as_user": "cwagent"
  }
```

CloudWatch agent configuration file: Metrics section

Fields common to Linux and Windows

On servers running either Linux or Windows Server, the `metrics` section includes the following fields:

- `namespace` – Optional. The namespace to use for the metrics collected by the agent. The default value is `CWAgent`. The maximum length is 255 characters. The following is an example:

  ```
  {
    "metrics": {
      "namespace": "Development/Product1Metrics",
     ......
     },
  }
  ```

- `append_dimensions` – Optional. Adds Amazon EC2 metric dimensions to all metrics collected by the agent. This also causes the agent to not publish the hostname as a dimension.

  The only supported key-value pairs for `append_dimensions` are shown in the following list. Any other key-value pairs are ignored. The agent supports these key-value pairs exactly as they are shown in the following list. You can't change the key values to publish different dimension names for them.

- `"ImageId":"${aws:ImageId}"` sets the instance's AMI ID as the value of the `ImageId` dimension.
- `"InstanceId":"${aws:InstanceId}"` sets the instance's instance ID as the value of the `InstanceId` dimension.
- `"InstanceType":"${aws:InstanceType}"` sets the instance's instance type as the value of the `InstanceType` dimension.
- `"AutoScalingGroupName":"${aws:AutoScalingGroupName}"` sets the instance's Auto Scaling group name as the value of the `AutoScalingGroupName` dimension.

If you want to append dimensions to metrics with arbitrary key-value pairs, use the `append_dimensions` parameter in the field for that particular type of metric.

If you specify a value that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see Amazon Elastic Compute Cloud (Amazon EC2) in the *Amazon Web Services General Reference*.

- `aggregation_dimensions` – Optional. Specifies the dimensions that collected metrics are to be aggregated on. For example, if you roll up metrics on the `AutoScalingGroupName` dimension, the metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

  You can roll up metrics along single or multiple dimensions. For example, specifying `[["InstanceId"], ["InstanceType"], ["InstanceId","InstanceType"]]` aggregates metrics for instance ID singly, instance type singly, and for the combination of the two dimensions.

  You can also specify `[]` to roll up all metrics into one collection, disregarding all dimensions.

- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends metrics. Specifying this and setting a private link enables you to send the metrics to an Amazon VPC endpoint. For more information, see What Is Amazon VPC?.

  The value of `endpoint_override` must be a string that is a URL.

  For example, the following part of the metrics section of the configuration file sets the agent to use a VPC Endpoint when sending metrics.

```
{
```

```
  "metrics": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXX.monitoring.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- `metrics_collected` – Required. Specifies which metrics are to be collected, including custom metrics collected through `StatsD` or `collectd`. This section includes several subsections.

  The contents of the `metrics_collected` section depend on whether this configuration file is for a server running Linux or Windows Server.

- `metrics_destinations` – Optional. Specifies one or more destinations for all metrics defined in `metrics_collected`. If specified here, it overrides the default destination of `cloudwatch`.
  - `cloudwatch` – Amazon CloudWatch.
  - amp – Amazon Managed Service for Prometheus.
    - `workspace_id` – The ID corresponding to the Amazon Managed Service for Prometheus workspace.

```
{
  "metrics": {
    "metrics_destinations": {
      "cloudwatch": {},
      "amp": {
        "workspace_id": "ws-abcd1234-ef56-7890-ab12-example"
      }
    }
  }
}
```

- `force_flush_interval` – Specifies in seconds the maximum amount of time that metrics remain in the memory buffer before being sent to the server. No matter the setting for this, if the size of the metrics in the buffer reaches 1 MB or 1000 different metrics, the metrics are immediately sent to the server.

  The default value is 60.

- `credentials` – Specifies an IAM role to use when sending metrics to a different account. If specified, this field contains one parameter, `role_arn`.

- `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending metrics to a different account. For more information, see Sending metrics, logs, and traces to a different account. If specified here, this value overrides the `role_arn` specified in the `agent` section of the configuration file, if any.
- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.

Linux section

On servers running Linux, the `metrics_collected` section of the configuration file can also contain the following fields.

Many of these fields can include a `measurement` sections that lists the metrics you want to collect for that resource. These `measurement` sections can either specify the complete metric name such as `swap_used`, or just the part of the metric name that will be appended to the type of resource. For example, specifying `reads` in the `measurement` section of the `diskio` section causes the `diskio_reads` metric to be collected.

- `collectd` – Optional. Specifies that you want to retrieve custom metrics using the `collectd` protocol. You use `collectd` software to send the metrics to the CloudWatch agent. For more information about the configuration options available for collectd, see Retrieve custom metrics with collectd.
- `cpu` – Optional. Specifies that CPU metrics are to be collected. This section is valid only for Linux instances. You must include at least one of the `resources` and `totalcpu` fields for any CPU metrics to be collected. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `resources` – Optional. Specify this field with a value of `*` to cause per-cpu metrics to be collected. The only allowed value is `*`.
- `totalcpu` – Optional. Specifies whether to report cpu metrics aggregated across all cpu cores. The default is true.
- `measurement` – Specifies the array of cpu metrics to be collected. Possible values are `time_active`, `time_guest`, `time_guest_nice`, `time_idle`, `time_iowait`, `time_irq`, `time_nice`, `time_softirq`, `time_steal`, `time_system`, `time_user`, `usage_active`, `usage_guest`, `usage_guest_nice`, `usage_idle`, `usage_iowait`, `usage_irq`, `usage_nice`, `usage_softirq`, `usage_steal`, `usage_system`, and `usage_user`. This field is required if you include `cpu`.

  By default, the unit for `cpu_usage_*` metrics is `Percent`, and `cpu_time_*` metrics don't have a unit.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `metrics_collection_interval` – Optional. Specifies how often to collect the cpu metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
- `append_dimensions` – Optional. Additional dimensions to use for only the cpu metrics. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics that the agent collects.

- `disk` – Optional. Specifies that disk metrics are to be collected. Collects metrics only for mounted volumes. This section is valid only for Linux instances. This section can include the following fields:

- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
- `resources` – Optional. Specifies an array of disk mount points. This field limits CloudWatch to collect metrics from only the listed mount points. You can specify `*` as the value to collect metrics from all mount points. The default value is to collect metrics from all mount points.
- `measurement` – Specifies the array of disk metrics to be collected. Possible values are `free`, `total`, `used`, `used_percent`, `inodes_free`, `inodes_used`, and `inodes_total`. This field is required if you include `disk`.

  > **Note**
  >
  > The `disk` metrics have a dimension for `Partition`, which means that the number of custom metrics generated is dependent on the number of partitions associated with your instance. The number of disk partitions you have depends on which AMI you are using and the number of Amazon EBS volumes you attach to the server.

  To see the default units for each `disk` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `ignore_file_system_types` – Specifies file system types to exclude when collecting disk metrics. Valid values include `sysfs`, `devtmpfs`, and so on.
- `drop_device` – Setting this to `true` causes `Device` to not be included as a dimension for disk metrics.

Preventing `Device` from being used as a dimension can be useful on instances that use the Nitro system because on those instances the device names change for each disk mount when the instance is rebooted. This can cause inconsistent data in your metrics and cause alarms based on these metrics to go to `INSUFFICIENT DATA` state.

The default is `false`.

- `metrics_collection_interval` – Optional. Specifies how often to collect the disk metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.

- `append_dimensions` – Optional. Specify key-value pairs to use as additional dimensions for only the disk metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

  One key-value pair that you can use is the following. You can also specify other custom key-value pairs.

  - `"VolumeId":"${aws:VolumeId}"` adds a `VolumeId` dimension to your block device disk metrics. For Amazon EBS volumes, this will be the Amazon EBS Volume ID. For EC2 instance store, this will be the device serial. Using this requires the `drop_device` parameter to be set to `false`.

- `diskio` – Optional. Specifies that disk i/o metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `resources` – Optional. If you specify an array of devices, CloudWatch collects metrics from only those devices. Otherwise, metrics for all devices are collected. You can also specify * as the value to collect metrics from all devices.
- `measurement` – Specifies the array of diskio and AWS NVMe driver metrics to be collected for Amazon EBS volumes and instance store volumes attached to Amazon EC2 instances. Possible diskio values are `reads`, `writes`, `read_bytes`, `write_bytes`, `read_time`, `write_time`, `io_time`, and `iops_in_progress`. For a list of the NVMe driver metrics for Amazon EBS volumes and Amazon EC2 instance store volumes, see Collect Amazon EBS NVMe driver metrics and Collect Amazon EC2 instance store volume NVMe driver metrics. This field is required if you include `diskio`.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

  For information on default units and description of the metrics, see Collect Amazon EBS NVMe driver metrics.
- `metrics_collection_interval` – Optional. Specifies how often to collect the diskio metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
- `append_dimensions` – Optional. Additional dimensions to use for only the diskio metrics. If you specify this field, it is used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `swap` – Optional. Specifies that swap memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value

of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of swap metrics to be collected. Possible values are `free`, `used`, and `used_percent`. This field is required if you include `swap`.

  To see the default units for each `swap` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `metrics_collection_interval` – Optional. Specifies how often to collect the swap metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

- `append_dimensions` – Optional. Additional dimensions to use for only the swap metrics. If you specify this field, it is used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent. It's collected as a high-resolution metric.

- `mem` – Optional. Specifies that memory metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter

don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of memory metrics to be collected. Possible values are `active`, `available`, `available_percent`, `buffered`, `cached`, `free`, `inactive`, `shared`, `total`, `used`, and `used_percent`. This field is required if you include `mem`.

  To see the default units for each `mem` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `metrics_collection_interval` – Optional. Specifies how often to collect the mem metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.

- `append_dimensions` – Optional. Additional dimensions to use for only the mem metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics that the agent collects.

- `net` – Optional. Specifies that networking metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated

metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `resources` – Optional. If you specify an array of network interfaces, CloudWatch collects metrics from only those interfaces. Otherwise, metrics for all devices are collected. You can also specify `*` as the value to collect metrics from all interfaces.
- `measurement` – Specifies the array of networking metrics to be collected. Possible values are `bytes_sent`, `bytes_recv`, `drop_in`, `drop_out`, `err_in`, `err_out`, `packets_sent`, and `packets_recv`. This field is required if you include `net`.

  To see the default units for each `net` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `metrics_collection_interval` – Optional. Specifies how often to collect the net metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
- `append_dimensions` – Optional. Additional dimensions to use for only the net metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `netstat` – Optional. Specifies that TCP connection state and UDP connection metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value

of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of netstat metrics to be collected. Possible values are `tcp_close`, `tcp_close_wait`, `tcp_closing`, `tcp_established`, `tcp_fin_wait1`, `tcp_fin_wait2`, `tcp_last_ack`, `tcp_listen`, `tcp_none`, `tcp_syn_sent`, `tcp_syn_recv`, `tcp_time_wait`, and `udp_socket`. This field is required if you include `netstat`.

  To see the default units for each `netstat` metric, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `metrics_collection_interval` – Optional. Specifies how often to collect the netstat metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information about high-resolution metrics, see High-resolution metrics.
- `append_dimensions` – Optional. Additional dimensions to use for only the netstat metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `processes` – Optional. Specifies that process metrics are to be collected. This section is valid only for Linux instances. This section can include the following fields:
  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can

specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of processes metrics to be collected. Possible values are `blocked`, `dead`, `idle`, `paging`, `running`, `sleeping`, `stopped`, `total`, `total_threads`, `wait`, and `zombies`. This field is required if you include `processes`.

  For all `processes` metrics, the default unit is None.

  Within the entry for each individual metric, you might optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of None for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.
- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

- `nvidia_gpu` – Optional. Specifies that NVIDIA GPU metrics are to be collected. This section is valid only for Linux instances on hosts that are configured with a NVIDIA GPU accelerator and have the NVIDIA System Management Interface (nvidia-smi) installed.

  The NVIDIA GPU metrics that are collected are prefixed with the string `nvidia_smi_` to distinguish them from the metrics collected for other accelerator types. This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent

sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

- `measurement` – Specifies the array of NVIDIA GPU metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect NVIDIA GPU metrics.

  Within the entry for each individual metric, you can optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit of `None` for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.
- `metrics_collection_interval` – Optional. Specifies how often to collect the NVIDIA GPU metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

- `jmx` – Optional. Specifies that you want to retrieve Java Management Extensions (JMX) metrics from the instance. For more information about the parameters that you can use in this section and the metrics that you can collect, see Collect Java Management Extensions (JMX) metrics.
- `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.
- `procstat` – Optional. Specifies that you want to retrieve metrics from individual processes. For more information about the configuration options available for procstat, see Collect process metrics with the procstat plugin.
- `statsd` – Optional. Specifies that you want to retrieve custom metrics using the `StatsD` protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard `StatsD` client to send the metrics to the CloudWatch agent. For more information about the configuration options available for StatsD, see Retrieve custom metrics with StatsD.
- `ethtool` – Optional. Specifies that you want to retrieve network metrics using the `ethtool` plugin. This plugin can import both the metrics collected by the standard ethtool utility, as well as network performance metrics from Amazon EC2 instances. For more information about the configuration options available for ethtool, see Collect network performance metrics.

- `efa` – Optional. Specifies that Elastic Fabric Adapter (EFA) metrics are to be collected. This section is valid only for Linux instances that have EFA devices attached. The agent collects EFA metrics from the `/sys/class/infiniband/` sysfs interface.

  This section can include the following fields:

  - `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.
  - `measurement` – Specifies the array of EFA metrics to be collected. This field is required. If you don't specify this field, the agent returns an error. You can specify metric names with or without the `efa_` prefix. For example, both `efa_tx_bytes` and `tx_bytes` are valid.

    For a list of the possible values to use here, see the **Metric** column in the table in Collect EFA metrics.
  - `metrics_collection_interval` – Optional. Specifies how often to collect the EFA metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file. The default value is 60 seconds.

  Each EFA metric is emitted with `device`, `port`, and `eniId` dimensions. To append the `InstanceId` dimension, use the `append_dimensions` field in the `metrics` section.

The following is an example of a `metrics` section for a Linux server. In this example, three CPU metrics, three netstat metrics, three process metrics, and one disk metric are collected, and the agent is set up to receive additional metrics from a `collectd` client.

```
"metrics": {
    "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId",
 "InstanceType"],[]],
    "metrics_collected": {
      "collectd": {},
      "cpu": {
        "resources": [
          "*"
```

```
        ],
        "measurement": [
          {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit": "Percent"},
          {"name": "cpu_usage_nice", "unit": "Percent"},
          "cpu_usage_guest"
        ],
        "totalcpu": false,
        "drop_original_metrics": [ "cpu_usage_guest" ],
        "metrics_collection_interval": 10,
        "append_dimensions": {
          "test": "test1",
          "date": "2017-10-01"
        }
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_syn_sent",
          "tcp_close"
        ],
        "metrics_collection_interval": 60
      },
       "disk": {
        "measurement": [
          "used_percent"
        ],
        "resources": [
          "*"
        ],
        "drop_device": true
      },
      "processes": {
        "measurement": [
          "running",
          "sleeping",
          "dead"
        ]
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
```

```
    }
  }
```

Windows Server

In the `metrics_collected` section for Windows Server, you can have subsections for each Windows performance object, such as `Memory`, `Processor`, and `LogicalDisk`. For information about what objects and counters are available, see Performance Counters in the Microsoft Windows documentation.

Within the subsection for each object, you specify a `measurement` array of the counters to collect. The `measurement` array is required for each object that you specify in the configuration file. You can also specify a `resources` field to name the instances to collect metrics from. You can also specify `*` for `resources` to collect separate metrics for every instance, excluding the `_Total` instance. To collect a single value aggregated across all instances, specify `_Total` for `resources`. If you omit `resources` for counters that have instances, such as `Processor`, the counters are not collected by the CloudWatch agent. If you omit `resources` for counters that don't have instances, such as `Memory`, the counters are collected. To determine whether counters have instances, you can use one of the following commands.

Powershell:

```
Get-Counter -ListSet *
```

Command line (not Powershell):

```
TypePerf.exe -q
```

Within each object section, you can also specify the following optional fields:

- `metrics_collection_interval` – Optional. Specifies how often to collect the metrics for this object, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.

- `append_dimensions` – Optional. Specifies additional dimensions to use for only the metrics for this object. If you specify this field, it's used in addition to dimensions specified in the global `append_dimensions` field that is used for all types of metrics collected by the agent.
- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

Within each counter section, you can also specify the following optional fields:

- `rename` – Specifies a different name to be used in CloudWatch for this metric.
- `unit` – Specifies the unit to use for this metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

There are other optional sections that you can include in `metrics_collected`:

- `statsd` – Enables you to retrieve custom metrics using the `StatsD` protocol. The CloudWatch agent acts as a daemon for the protocol. You use any standard `StatsD` client to send the metrics to the CloudWatch agent. For more information, see Retrieve custom metrics with StatsD.
- `procstat` – Enables you to retrieve metrics from individual processes. For more information, see Collect process metrics with the procstat plugin.
- `jmx` – Optional. Specifies that you want to retrieve Java Management Extensions (JMX) metrics from the instance. For more information about the fields that you can use in this section and the metrics that you can collect, see Collect Java Management Extensions (JMX) metrics.
- `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.

The following is an example `metrics` section for use on Windows Server. In this example, many Windows metrics are collected, and the computer is also set to receive additional metrics from a `StatsD` client.

```
"metrics": {
    "metrics_collected": {
      "statsd": {},
      "Processor": {
        "measurement": [
          {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
          "% Interrupt Time",
          "% User Time",
          "% Processor Time"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      },
      "LogicalDisk": {
        "measurement": [
          {"name": "% Idle Time", "unit": "Percent"},
          {"name": "% Disk Read Time", "rename": "DISK_READ"},
          "% Disk Write Time"
        ],
        "resources": [
          "*"
        ]
      },
      "Memory": {
        "metrics_collection_interval": 5,
        "measurement": [
          "Available Bytes",
          "Cache Faults/sec",
          "Page Faults/sec",
          "Pages/sec"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "Network Interface": {
        "metrics_collection_interval": 5,
        "measurement": [
```

```
          "Bytes Received/sec",
          "Bytes Sent/sec",
          "Packets Received/sec",
          "Packets Sent/sec"
        ],
        "resources": [
          "*"
        ],
        "append_dimensions": {
          "d3": "win_bo"
        }
      },
      "System": {
        "measurement": [
          "Context Switches/sec",
          "System Calls/sec",
          "Processor Queue Length"
        ],
        "append_dimensions": {
          "d1": "win_foo",
          "d2": "win_bar"
        }
      }
    },
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}",
      "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
    },
    "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"], ["d1"],[]]
    }
  }
```

CloudWatch agent configuration file: Logs section

The `logs` section includes the following fields:

- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.

- `backpressure_mode` – Optional. Specifies the behavior when the CloudWatch agent is ingesting logs faster than it can send to CloudWatch Logs, resulting in backpressure. Backpressure can happen from network issues, API throttling, or high log volume.

  The agent supports the following values:

  - `fd_release` – Releases file descriptors for deleted files during backpressure conditions. This option can help prevent disk space exhaustion when external log rotation or cleanup processes remove files while the agent maintains open file descriptors. The `auto_removal` option takes precedence over the `backpressure_mode` option being set to `fd_release`. When `auto_removal` is enabled, the CloudWatch agent processes the file to completion without releasing the file descriptor.

    > **Important**
    >
    > Using `fd_release` can result in the CloudWatch agent being unable to read log files to completion, causing log loss.

- `concurrency` – Optional. Specifies the number of shared log publishers used to concurrently publish log files to CloudWatch Logs.

  If you omit this field, each log file destination (log group, stream combination) has a single shared log publisher, which can lead to bottlenecks for large files or when writing multiple files to the same destination. Enabling concurrency can help with throughput.

- `logs_collected` – Required if the `logs` section is included. Specifies which log files, Windows event logs, and journald logs the agent collects from the server. It can include the following fields: `files`, `windows_events` (Windows only), and `journald` (Linux only).
  - `files` – Specifies which regular log files the CloudWatch agent is to collect. It contains one field, `collect_list`, which further defines these files.
    - `collect_list` – Required if `files` is included. Contains an array of entries, each of which specifies one log file to collect. Each of these entries can include the following fields:
      - `file_path` – Specifies the path of the log file to upload to CloudWatch Logs. Standard Unix glob matching rules are accepted, with the addition of `**` as a *super asterisk*. For example, specifying `/var/log/**.log` causes all `.log` files in the `/var/log` directory tree to be collected. For more examples, see Glob Library.

You can also use the standard asterisk as a standard wildcard. For example, `/var/log/system.log*` matches files such as `system.log_1111`, `system.log_2222`, and so on in `/var/log`.

Only the latest file is pushed to CloudWatch Logs based on file modification time. We recommend that you use wildcards to specify a series of files of the same type, such as `access_log.2018-06-01-01` and `access_log.2018-06-01-02`, but not multiple kinds of files, such as `access_log_80` and `access_log_443`. To specify multiple kinds of files, add another log stream entry to the agent configuration file so that each kind of log file goes to a different log stream.

- `auto_removal` – Optional. If this is `true`, the CloudWatch agent automatically deletes this log file after reading it and it has been rotated. Usually the log files are deleted after their entire contents are uploaded to CloudWatch Logs. However, if the agent reaches the EOF (end of file) and also detects another newer log file that matches the same `file_path`, the agent deletes the OLD file, so you must make sure that you are done writing to the OLD file before creating the NEW file. The RUST tracing library has a known incompatibility because it will potentially create a NEW log file and then still attempt to write to the OLD log file.

  The agent only removes complete files from logs that create multiple files, such as logs that create separate files for each date. If a log continuously writes to a single file, it is not removed.

  If you already have a log file rotation or removal method in place, we recommend that you omit this field or set it to `false`.

  If you omit this field, the default value of `false` is used.

- `log_group_name` – Optional. Specifies what to use as the log group name in CloudWatch Logs.

  We recommend that you use this field to specify a log group name to prevent confusion. If you omit `log_group_name`, the value of `file_path` up to the final dot is used as the log group name. For example, if the file path is `/tmp/TestLogFile.log.2017-07-11-14`, the log group name is `/tmp/TestLogFile.log`.

  If you specify a log group name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}`

retrieves the hostname from the EC2 metadata, and {local_hostname} uses the hostname from the network configuration file.

If you use these variables to create many different log groups, keep in mind the limit of 1,000,000 log groups per Region per account.

Allowed characters include a–z, A–Z, 0–9, '_' (underscore), '-' (hyphen), '/' (forward slash), and '.' (period).

- log_group_class – Optional. Specifies which log group class to use for the new log group. For more information about log group classes, see Log classes.

  Valid values are STANDARD and INFREQUENT_ACCESS. If you omit this field, the default of STANDARD is used.

  > **Important**
  >
  > After a log group is created, its class can't be changed.

- log_stream_name – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use {instance_id}, {hostname}, {local_hostname}, and {ip_address} as variables within the name. {hostname} retrieves the hostname from the EC2 metadata, and {local_hostname} uses the hostname from the network configuration file.

  If you omit this field, the value of the log_stream_name parameter in the global logs section is used. If that is also omitted, the default value of {instance_id} is used.

  If a log stream doesn't already exist, it's created automatically.

- retention_in_days – Optional. Specifies the number of days to retain the log events in the specified log group.
  - If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.
  - If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

    The CloudWatch agent wizard uses -1 as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention.

This `-1` value set by the wizard specifies that the events in the log group will never expire. However, manually editing this value to `-1` has no effect.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

> **Note**
>
> The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies.

> **Warning**
>
> If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `filters` – Optional. Can contain an array of entries, each of which specifies a regular expression and a filter type to specify whether to publish or drop log entries that match the filter. If you omit this field, all logs in the log file are published to CloudWatch Logs. If you include this field, the agent processes each log message with all of the filters that you specify, and only the log events that pass all of the filters are published to CloudWatch Logs. The log entries that don't pass all of the filters will still remain in the host's log file, but will not be sent to CloudWatch Logs.

  Each entry in the filters array can include the following fields:

  - `type`– Denotes the type of filter. Valid values are `include` and `exclude`. With `include`, the log entry must match the expression to be published to CloudWatch Logs. With `exclude`, each log entry that matches the filter is not sent to CloudWatch Logs.

- `expression`– A regular expression string that follows the RE2 Syntax.

> **Note**
>
> The CloudWatch agent doesn't check the performance of any regular expression that you supply, or restrict the run time of the evaluation of the regular expressions. We recommend that you are careful not to write an expression that is expensive to evaluate. For more information about possible issues, see Regular expression Denial of Service - ReDoS

For example, the following excerpt of the CloudWatch agent configuration file publishes logs that are PUT and POST requests to CloudWatch Logs, but excluding logs that come from Firefox.

```
"collect_list": [
  {
    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
    "log_group_name": "test.log",
    "log_stream_name": "test.log",
    "filters": [
      {
        "type": "exclude",
        "expression": "Firefox"
      },
      {
        "type": "include",
        "expression": "P(UT|OST)"
      }
    ]
  },
  .....
]
```

> **Note**
>
> The order of the filters in the configuration file matters for performance. In the preceding example, the agent drops all the logs that match `Firefox` before it starts evaluating the second filter. To cause fewer log entries to be evaluated by

more than one filter, put the filter that you expect to rule out more logs first in the configuration file.

- `timezone` – Optional. Specifies the time zone to use when putting timestamps on log events. The valid values are `UTC` and `Local`. The default value is `Local`.

  This parameter is ignored if you don't specify a value for `timestamp_format`.

- `timestamp_format` – Optional. Specifies the timestamp format, using plaintext and special symbols that start with %. If you omit this field, the current time is used. If you use this field, you can use the symbols in the following list as part of the format.

  Note

  This parameter is not considered when the `file_path` is set to `amazon-cloudwatch-agent.log`

  If a single log entry contains two time stamps that match the format, the first time stamp is used.

  This list of symbols is different than the list used by the older CloudWatch Logs agent. For a summary of these differences, see Timestamp differences between the CloudWatch agent and the earlier CloudWatch Logs agent

  %y

  Year without century as a zero-padded decimal number. For example, 19 to represent 2019.

  %Y

  Year with century as a decimal number. For example, 2019.

  %b

  Month as the locale's abbreviated name

  %B

  Month as the locale's full name

`%m`

Month as a zero-padded decimal number

`%-m`

Month as a decimal number (not zero-padded)

`%d`

Day of the month as a zero-padded decimal number

`%-d`

Day of the month as a decimal number (not zero-padded)

`%A`

Full name of weekday, such as `Monday`

`%a`

Abbreviation of weekday, such as `Mon`

`%H`

Hour (in a 24-hour clock) as a zero-padded decimal number

`%I`

Hour (in a 12-hour clock) as a zero-padded decimal number

`%-I`

Hour (in a 12-hour clock) as a decimal number (not zero-padded)

`%p`

AM or PM

`%M`

Minutes as a zero-padded decimal number

`%-M`

Minutes as a decimal number (not zero-padded)

`%S`

Seconds as a zero-padded decimal number

`%-S`

Seconds as a decimal number (not zero padded)

`%f`

Fractional seconds as a decimal number (1-9 digits), zero-padded on the left.

`%Z`

Time zone, for example PST

`%z`

Time zone, expressed as the offset between the local time zone and UTC. For example, `-0700`. Only this format is supported. For example, `-07:00` isn't a valid format.

- `multi_line_start_pattern` – Specifies the pattern for identifying the start of a log message. A log message is made of a line that matches the pattern and any subsequent lines that don't match the pattern.

  If you omit this field, multi-line mode is disabled, and any line that begins with a non-whitespace character closes the previous log message and starts a new log message.

  If you include this field, you can specify `{timestamp_format}` to use the same regular expression as your timestamp format. Otherwise, you can specify a different regular expression for CloudWatch Logs to use to determine the start lines of multi-line entries.
- `encoding` – Specified the encoding of the log file so that it can be read correctly. If you specify an incorrect coding, there might be data loss because characters that can't be decoded are replaced with other characters.

  The default value is `utf-8`. The following are all possible values:

  ```
  ascii, big5, euc-jp, euc-kr, gbk, gb18030, ibm866, iso2022-jp,
  iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7,
  iso8859-8, iso8859-8-i, iso8859-10, iso8859-13, iso8859-14,
  iso8859-15, iso8859-16, koi8-r, koi8-u, macintosh, shift_jis, utf-8,
  utf-16, utf-16le, UTF-16, UTF-16LE, windows-874, windows-1250,
  ```

`windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, x-mac-cyrillic`

- `service.name` – Optional. Specifies the service name to be used to populate the entity for finding related telemetry.
- `deployment.environment` – Optional. Specifies the environment name to be used to populate the entity for finding related telemetry.
- `trim_timestamp` – Optional. If this is true, the CloudWatch agent will remove the timestamp matched by `timestamp_format` from the line before sending it to CloudWatch Logs. The LogEvent will still contain the `timestamp` field.

  If you omit this field, the default value of `false` is used.

- The `windows_events` section specifies the type of Windows events to collect from servers running Windows Server. It includes the following fields:
  - `collect_list` – Required if `windows_events` is included. Specifies the types and levels of Windows events to be collected. Each log to be collected has an entry in this section, which can include the following fields:
    - `event_name` – Specifies the type of Windows events to log. This is equivalent to the Windows event log channel name: for example, `System`, `Security`, `Application`, and so on. This field is required for each type of Windows event to log.

      > **Note**
      >
      > When CloudWatch retrieves messages from a Windows log channel, it looks up the log channel based on its `Full Name` property. Meanwhile, the Windows Event Viewer navigation pane displays the `Log Name` property of log channels. The `Full Name` and `Log Name` do not always match. To confirm the `Full Name` of a channel, right-click on it in the Windows Event viewer and open **Properties**.

    - `event_levels` – Optional. Specifies the levels of event to log. You must specify each level to log. Possible values include `INFORMATION`, `WARNING`, `ERROR`, `CRITICAL`, and `VERBOSE`. This field is optional for each type of Windows event to log. and can be used with other filtering option like `event_ids` and `filters`.
    - `event_ids` – Optional. Contains an array of Windows Event IDs to specify which events to collect from the Windows Event Log. When this field is excluded, all events from the

specified event log are collected. When this field is included, the agent only collects events that match the specified Event IDs.

Each entry in the `event_ids` array should be a numeric Event ID value and can be used with other filtering options. See the third entry in the config sample below.

> **Note**
>
> Using `event_ids` for filtering is recommended over regex expressions when you need to filter by Event ID, as it provides better performance.

- `filters` – Optional. Contains an array of entries. Each entry specifies a regular expression and a filter type to specify whether to publish or drop the log entries that match the filter. When the field is included, the agent processes each log message with all of the filters that you specify, and only the log events that pass all filters are published to CloudWatch Logs. The windows event logs that doesn't pass all of the filters will be dropped and not sent to CloudWatch Logs. The filters section can also be used with other filtering mechanisms like event ids [4624, 4625] and system levels (Information, Error, or Critical) to effectively filter logs and push to CloudWatch.

  Each entry in the filters array can include the following fields:

  - `type` – Specifies the type of filter. Valid values are `include` and `exclude`. With include, the windows events entry must match the expression to be published to CloudWatch Logs. With exclude, each windows event log entry that matches the filter is not sent to CloudWatch Logs.
  - `expression` – A regular expression string that follows the RE2 Syntax.

  > **Note**
  >
  > The CloudWatch agent does not validate regular expressions that you provide. It also does not limit their evaluation time. Write your expressions carefully to avoid performance issues. For more information about security risks, see Regular expression Denial of Service - ReDoS .

  In the example agent configuration below:

For the first entry, the agent pushes logs that contain database failure messages, any authentication related activities, and all login events (both successful and failed attempts) to CloudWatch. Any log that doesn't match this pattern is dropped.

In the second entry, initial filtering is done based on event ids for windows event subscription. The agent collects all logs that contain the string user, discarding logs that don't match these patterns. The agent then drops logs containing `successful` before sending the remaining logs to CloudWatch Logs. Every filter type is applied to each windows event log before sending to CloudWatch.

```
"collect_list": [
  {
        "event_name": "Application",
        "log_group_name": "ApplicationEvents",
        "log_stream_name": "ApplicationEvents",
        "filters": [
            {
                "type": "include",
                "expression": "Database.*failed|Authentication.*|login.*"
            }
        ]
    },
    {
        "event_name": "System",
        "log_group_name": "SystemEvents",
        "log_stream_name": "Logon-events",
        "event_ids": [
            4624,
            4625
         ],
        "filters": [
            {
                "type": "include",
                "expression": ".*user.*"
            },
            {
                "type": "exclude",
                "expression": ".*successful.*"
            }
         ]
     }
```

```
  .....
]
```

> **Note**
>
> The order of the filters in the configuration will impact the performance. In the second entry, the agent drops all the logs that does not match the user before it starts evaluating the second filter expression. For optimal performance, order filters from highest to lowest exclusion rate.

While you can filter out logs on event ids and system levels in the filter expression, it is recommended to use the `event_ids` and `log_level` as shown in the second entry for improved performance.

> **Warning**
>
> Even though all filtering mechanisms (event_levels, event_ids, filters) are optional, at least one is required during agent configuration to filter logs.

- `log_group_name` – Required. Specifies what to use as the log group name in CloudWatch Logs.
- `log_stream_name` – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. `{hostname}` retrieves the hostname from the EC2 metadata, and `{local_hostname}` uses the hostname from the network configuration file.

  If you omit this field, the value of the `log_stream_name` parameter in the global `logs` section is used. If that is also omitted, the default value of `{instance_id}` is used.

  If a log stream doesn't already exist, it's created automatically.
- `event_format` – Optional. Specifies the format to use when storing Windows events in CloudWatch Logs. `xml` uses the XML format as in Windows Event Viewer. `text` uses the legacy CloudWatch Logs agent format.
- `retention_in_days` – Optional. Specifies the number of days to retain the Windows events in the specified log group.

- If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire.
- If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

  The CloudWatch agent wizard uses `-1` as the default value for this field when it is used to create the agent configuration file and you don't specify a value for log retention. This `-1` value set by the wizard specifies that the events in the log group don't expire. However, manually editing this value to `-1` has no effect.

Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

If you configure the agent to write multiple log streams to the same log group, specifying the `retention_in_days` in one place will set the log retention for the entire log group. If you specify `retention_in_days` for the same log group in multiple places, the retention is set if all of those values are equal. However, if different `retention_in_days` values are specified for the same log group in multiple places, the log retention will not be set and the agent will stop, returning an error.

> **Note**
>
> The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies.

> **Warning**
>
> If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- Use the `journald` section to specify which systemd journal logs to collect from Linux servers. Use this section only for Linux instances. It includes the following fields:

- `collect_list` – Required if `journald` is included. Specifies which journal log entries the agent collects. Each entry in this section defines a set of journal logs to collect and can include the following fields:
  - `units` – Optional. Specifies an array of systemd unit names to filter journal entries. If you include this field, the agent collects only journal entries from the specified systemd units. If you omit this field, the agent collects journal entries from all units. The agent accepts wildcards for pattern matching. For example, `["sshd*"]` matches `sshd.service`, `sshd@1.service`, and any other unit starting with `sshd`. You can specify multiple units such as `["sshd", "nginx", "cloudwatch-agent"]` to collect logs from several services.
  - `priority` – Optional. Specifies the minimum journal priority level to collect. If you include this field, the agent collects journal entries at the specified priority level and all levels more severe. If you omit this field, the default is `info`, collecting `info` and above and excluding debug. Set `"priority": "debug"` to collect all levels. Valid values are `emerg`, `alert`, `crit`, `err`, `warning`, `notice`, `info`, and debug. For example, setting `"priority": "err"` collects entries at `emerg` (0), `alert` (1), `crit` (2), and `err` (3) levels, excluding `warning`, `notice`, `info`, and debug.

    The following table shows the journal priority levels and their numeric values.

Priority	Numeric Value	Description
`emerg`	0	System is unusable
`alert`	1	Action must be taken immediately
`crit`	2	Critical conditions
`err`	3	Error conditions
`warning`	4	Warning conditions
`notice`	5	Normal but significant conditions
`info`	6	Informational messages
`debug`	7	Debug-level messages

- `matches` – Optional. Specifies an array of journal field match objects to filter journal entries by metadata fields. Each object in the array represents a set of conditions. Fields within a single object are AND'd together, while separate objects in the array are OR'd against each other. If you include this field, the agent collects journal entries that match any one of the specified objects. If you omit this field, the agent applies no field-level filtering.

  For example, specifying `[{"_UID": "0"}]` collects only journal entries from the root user. Specifying `[{"_SYSTEMD_UNIT": "sshd.service"}, {"_SYSTEMD_UNIT": "kubelet.service", "_UID": "1000"}]` collects entries from the SSH daemon OR entries from kubelet that are also from UID 1000. You can combine `matches` with `units` and `priority` for compound filtering.

  > **Note**
  >
  > Journal field names are case-sensitive and typically use uppercase with an underscore prefix for trusted fields set by the journal (for example, `_UID`, `_PID`, `_SYSTEMD_UNIT`). User-defined fields do not have the underscore prefix.

- `filters` – Optional. Contains an array of entries. Each entry specifies a regular expression and a filter type to specify whether to publish or drop the log entries that match the filter. If you include this field, the agent processes each log message with all filters you specify, and only the log events that pass all filters are published to CloudWatch Logs. The journald logs that don't pass all of the filters are dropped and not sent to CloudWatch Logs. The filters section can also be used with other filtering mechanisms like `units` and `priority` to effectively filter logs and push to CloudWatch.

  Each entry in the filters array can include the following fields:

  - `type` – Specifies the type of filter. Valid values are `include` and `exclude`. With `include`, the journald entry must match the expression to be published to CloudWatch Logs. With `exclude`, each journald log entry that matches the filter is not sent to CloudWatch Logs.
  - `expression` – A regular expression string that follows the RE2 Syntax.

> **Note**
>
> The CloudWatch agent does not validate regular expressions that you provide. It also does not limit their evaluation time. Write your expressions carefully to avoid performance issues. For more information about security risks, see Regular expression Denial of Service - ReDoS.

> **Note**
>
> The order of the filters in the configuration will impact the performance. For optimal performance, order filters from highest to lowest exclusion rate.

- `retention_in_days` – Optional. Specifies the number of days to retain the log events in the specified log group. If the agent is creating this log group now, and you omit this field, the retention of this new log group is set to never expire. If this log group already exists and you specify this field, the new retention that you specify is used. If you omit this field for a log group that already exists, the log group's retention is not changed.

  Valid values are 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 2192, 2557, 2922, 3288, and 3653.

  > **Note**
  >
  > The agent's IAM role or IAM user must have the `logs:PutRetentionPolicy` for it to be able to set retention policies.

  > **Warning**
  >
  > If you set `retention_in_days` for a log group that already exists, all logs in that log group that were published before the number of days that you specify are deleted. For example, setting it to 3 would cause all logs from 3 days ago and before to be deleted.

- `log_group_name` – Required. Specifies what to use as the log group name in CloudWatch Logs. You can use `{instance_id}`, `{hostname}`, `{local_hostname}`,

and {ip_address} as variables within the name. {hostname} retrieves the hostname from the EC2 metadata, and {local_hostname} uses the hostname from the network configuration file.

- log_stream_name – Optional. Specifies what to use as the log stream name in CloudWatch Logs. As part of the name, you can use {instance_id}, {hostname}, {local_hostname}, and {ip_address} as variables within the name. If you omit this field, the value of the log_stream_name parameter in the global logs section is used. If that is also omitted, the default value of {instance_id} is used.

> **Note**
>
> SELinux support requires the amazon-cloudwatch-agent-selinux policy version 1.1.0 or newer.

> **Note**
>
> Running the agent as any user other than root requires the user to be added to the systemd-journal group to read the journal. On restart, the agent resumes from the last collected journal entry and does not collect historical data.

In the following example agent configuration:

- For the first entry, the agent collects all journal logs without any filtering and sends them to the journald-all-logs log group.
- In the second entry, the agent collects only logs from the sshd systemd unit.
- In the third entry, the agent collects journal entries at err priority and above.
- In the fourth entry, the agent uses compound filtering to collect logs from the systemd-logind unit that are also from the root user (_UID of 0).
- For the last entry, the agent collects journald logs from the sshd service and applies layered regex filtering.

```
"journald": {
    "collect_list": [
        {
            "log_group_name": "journald-all-logs",
```

```
            "log_stream_name": "{instance_id}",
            "retention_in_days": 7
        },
        {
            "log_group_name": "journald-sshd",
            "log_stream_name": "{instance_id}",
            "units": ["sshd"],
            "retention_in_days": 7
        },
        {
            "log_group_name": "journald-errors",
            "log_stream_name": "{instance_id}",
            "priority": "err",
            "retention_in_days": 7
        },
        {
            "log_group_name": "journald-root-logind",
            "log_stream_name": "{instance_id}",
            "units": ["systemd-logind"],
            "matches": [{"_UID": "0"}],
            "retention_in_days": 7
        },
        {
            "log_group_name": "journald-filters",
            "log_stream_name": "{instance_id}",
            "units": ["sshd"],
            "filters": [
                {
                    "type": "include",
                    "expression": ".*user.*"
                },
                {
                    "type": "exclude",
                    "expression": ".*successful.*"
                }
            ],
            "retention_in_days": 7
        }
    ]
}
```

> **Note**
>
> When you combine `units`, `priority`, and `matches` filters in a single entry, all filters are applied together. A journal entry must satisfy all specified filters to be collected and sent to CloudWatch Logs.

- `log_stream_name` – Optional. Specifies the default log stream name to be used for any logs or Windows events that don't have individual log stream names defined in the `log_stream_name` parameter within their entry in `collect_list`.
- `endpoint_override` – Specifies a FIPS endpoint or private link to use as the endpoint where the agent sends logs. Specifying this field and setting a private link enables you to send the logs to an Amazon VPC endpoint. For more information, see What Is Amazon VPC?.

  The value of `endpoint_override` must be a string that is a URL.

  For example, the following part of the logs section of the configuration file sets the agent to use a VPC Endpoint when sending logs.

```
{
  "logs": {
    "endpoint_override": "vpce-XXXXXXXXXXXXXXXXXXXXXXXX.logs.us-
east-1.vpce.amazonaws.com",
   ......
   },
}
```

- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5.

  If you are using the agent to report high-resolution metrics in embedded metric format, and you are setting alarms on those metrics, keep this parameter set to the default value of 5. Otherwise, the metrics are reported with a delay that can cause alarming on partial or incomplete data.
- `credentials` – Specifies an IAM role to use when sending logs to a different AWS account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending logs to a different AWS account. For more information, see Sending metrics, logs, and traces to a

different account. If specified here, this overrides the `role_arn` specified in the `agent` section of the configuration file, if any.

- `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.
  - `application_signals` (Optional) Specifies that you want to enable CloudWatch Application Signals For more information about this configuration, see Enable CloudWatch Application Signals.
  - `kubernetes` – This field can contain an `enhanced_container_insights` parameter, which you can use to enable Container Insights with enhanced observability for Amazon EKS.
    - `enhanced_container_insights` – Set this to `true` to enable Container Insights with enhanced observability for Amazon EKS. For more information, see Amazon EKS.
    - `accelerated_compute_metrics` – Set this to `false` to opt out of collecting Nvidia GPU metrics on Amazon EKS clusters. For more information, see NVIDIA GPU metrics.
  - `emf` – To collect metrics embedded in logs, it is no longer necessary to add this `emf` field. This is a legacy field that specified that the agent is to collect logs that are in embedded metric format. You can generate metric data from these logs. For more information, see Embedding metrics within logs.
  - `otlp` – Optional. Specifies that you want to collect metrics from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry.

The following is an example of a `logs` section.

```
"logs":{
    "logs_collected": {
    "files": {
            "collect_list": [
                   {
                        "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\amazon-cloudwatch-agent.log",
                       "log_group_name": "amazon-cloudwatch-agent.log",
                       "log_stream_name": "my_log_stream_name_1"
                   },
                   {
                       "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\Logs\\test.log",
```

```
                    "log_group_name": "test.log",
                    "log_stream_name": "my_log_stream_name_2"
                }
            ]
        },
    "windows_events": {
             "collect_list": [
                            {
                    "event_name": "System",
                    "event_ids": [
                        1001,
                        1008
                    ],
                    "log_group_name": "System",
                    "log_stream_name": "System"
                },
                {
                    "event_name": "CustomizedName",
                    "event_levels": [
                        "INFORMATION",
                        "ERROR"
                    ],
                    "log_group_name": "CustomizedLogGroup",
                    "log_stream_name": "CustomizedLogStream"
                },
                {
                    "event_name": "Application",
                    "event_levels": [
                        "INFORMATION",
                        "ERROR"
                    ],
                    "event_ids":[
                         7369,
                         5624
                    ],
                    "log_group_name": "CustomizedLogGroup",
                    "log_stream_name": "CustomizedLogStream"
                }
            ]
        }
     },
     "log_stream_name": "my_log_stream_name",
     "metrics_collected": {
      "kubernetes": {
```

```
        "enhanced_container_insights": true
      }
    }
  }
```

CloudWatch agent configuration file: Traces section

By adding a `traces` section to the CloudWatch agent configuration file, you can enable CloudWatch Application Signals or collect traces from X-Ray and from the OpenTelemetry instrumentation SDK and send them to X-Ray.

> **Important**
>
> The agent's IAM role or IAM user must have the **AWSXrayWriteOnlyAccess** policy to send trace data to X-Ray.

For a quick start for collecting traces, you can add just the following to the CloudWatch agent configuration file.

```
"traces_collected": {
        "xray": {
        },
        "otlp": {
        }
      }
```

If you add the previous section to the CloudWatch agent configuration file and restart the agent, this causes the agent to start collecting traces using the following default options and values. For more information about these parameters, see the parameter definitions later in this section.

```
"traces_collected": {
        "xray": {
            "bind_address": "127.0.0.1:2000",
            "tcp_proxy": {
              "bind_address": "127.0.0.1:2000"
            }
        },
        "otlp": {
            "grpc_endpoint": "127.0.0.1:4317",
            "http_endpoint": "127.0.0.1:4318"
```

```
        }
      }
```

The `traces` section can include the following fields:

- `traces_collected` – Required if the `traces` section is included. Specifies which SDKs to collect traces from. It can include the following fields:
  - `application_signals` – Optional. Specifies that you want to enable CloudWatch Application Signals For more information about this configuration, see Enable CloudWatch Application Signals.
  - `xray` – Optional. Specifies that you want to collect traces from the X-Ray SDK. This section can include the following fields:
    - `bind_address` – Optional. Specifies the UDP address for the CloudWatch agent to use to listen for X-Ray traces. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

      If you omit this field, the default of `127.0.0.1:2000` is used.
    - `tcp_proxy` – Optional. Configures the address for a proxy used to support X-Ray remote sampling. For more information, see Configuring sampling rules in the X-Ray documentation.

      This section can contain the following field.
      - `bind_address` – Optional. Specifies the TCP address to which the CloudWatch agent should set up the proxy. The format is `ip:port`. This address must match the address set in the X-Ray SDK.

        If you omit this field, the default of `127.0.0.1:2000` is used.
  - `otlp` – Optional. Specifies that you want to collect traces from the OpenTelemetry SDK. For more information about the fields that you can use in this section, see Collect metrics and traces with OpenTelemetry). For more information about the AWS Distro for OpenTelemetry, see AWS Distro for OpenTelemetry. For more information about the AWS Distro for OpenTelemetry SDKs, see Introduction.

    This section can include the following fields:
    - `grpc_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry traces sent using gRPC Remote Procedure Calls. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK.

If you omit this field, the default of `127.0.0.1:4317` is used.

- `http_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OTLP traces sent over HTTP. The format is `ip:port`. This address must match the address set for the HTTP exporter in the OpenTelemetry SDK.

  If you omit this field, the default of `127.0.0.1:4318` is used.

- `concurrency` – Optional. Specifies the maximum number of concurrent calls to X-Ray that can be used to upload traces. The default value is 8
- `local_mode` – Optional. If `true`, the agent doesn't collect Amazon EC2 instance metadata. The default is `false`
- `endpoint_override` – Optional. Specifies a FIPS endpoint or private link to use as the endpoint where the CloudWatch agent sends traces. Specifying this field and setting a private link enables you to send the traces to an Amazon VPC endpoint. For more information, see What is Amazon VPC

  The value of `endpoint_override` must be a string that is a URL.
- `region_override` – Optional. Specifies the Region to use for the X-Ray endpoint. The CloudWatch agent sends the traces to X-Ray in the specified Region. If you omit this field, the agent sends the traces to the Region where the Amazon EC2 instance is located.

  If you specify a Region here, it takes precedence over the setting of the `region` parameter in the `agent` section of the configuration file.
- `proxy_override` – Optional. Specifies the proxy server address for the CloudWatch agent to use when sending requests to X-Ray. The proxy server's protocol must be specified as part of this address.
- `credentials` – Specifies an IAM role to use when sending traces to a different AWS account. If specified, this field contains one parameter, `role_arn`.
  - `role_arn` – Specifies the ARN of an IAM role to use for authentication when sending traces to a different AWS account. For more information, see Sending metrics, logs, and traces to a different account. If specified here, this overrides the `role_arn` specified in the `agent` section of the configuration file, if any.
- `transit_spans_in_otlp_format` – Optional. If `true`, sends traces to X-Ray in the OpenTelemetry Protocol format, which supports span events in Transaction Search. For more information, see Adding custom attributes. The default is `false`.

CloudWatch agent configuration file: OpenTelemetry section

The `opentelemetry` section depends on the agent's OTLP receiver, which was introduced in CloudWatch agent version 1.300070.0. Earlier versions do not support it. To check the installed version, read `/opt/aws/amazon-cloudwatch-agent/bin/CWAGENT_VERSION`.

> **Note**
>
> On Amazon Linux 2023, the `yum install amazon-cloudwatch-agent` and `dnf install amazon-cloudwatch-agent` commands install a version that meets this minimum. If your package repository offers a version earlier than 1.300070.0, install the agent from the CloudWatch agent download instead. For more information, see Collect metrics, logs, and traces using the CloudWatch agent.

The `opentelemetry` section can include the following fields:

- `cluster_name` – Optional. Specifies the name of the Kubernetes cluster. The agent applies this value to all telemetry collected under the `opentelemetry` section.
- `resource_attributes` – Optional. Specifies additional resource attributes to add to every metric, log, and trace collected under the `opentelemetry` section. This field is an object of string key-value pairs. You can specify up to 30 attributes. The maximum length of each value is 1024 characters. For attributes that the agent detects automatically, such as `cloud.region`, `host.id`, and `cloud.provider`, the detected value takes precedence. Use this field for attributes that the agent does not detect on its own.
- `collect` – Required. Specifies which telemetry sources the agent collects. You must include at least one of the following fields:
  - `otlp` – Optional. Specifies that the agent receives metrics, traces, and logs sent to it over the OpenTelemetry Protocol (OTLP). This section can include the following fields:
    - `grpc_endpoint` – Optional. Specifies the address the agent uses to listen for OTLP data sent using gRPC. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK. If you omit this field, the agent uses the default `127.0.0.1:4317`. In a containerized deployment, set this to `0.0.0.0:4317` so the agent accepts traffic from other pods.
    - `http_endpoint` – Optional. Specifies the address the agent uses to listen for OTLP data sent over HTTP. The format is `ip:port`. This address must match the address set for the HTTP exporter in the OpenTelemetry SDK. If you omit this field, the agent uses the default

`127.0.0.1:4318`. In a containerized deployment, set this to `0.0.0.0:4318` so the agent accepts traffic from other pods.

- `span_metrics_enabled` – Optional. If `true`, the agent derives request, error, and duration metrics from received traces. The default value is `false`.

- `files` – Optional. Specifies that the agent collects log files from the host. It contains one field, `collect_list`.
  - `collect_list` – Required if `files` is included. Contains an array of entries, each of which specifies one set of log files to collect. The array must contain at least one entry. Each entry can include the following fields:
    - `file_path` – Required. Specifies the path or glob pattern of the log files to collect. The maximum length is 4096 characters.
    - `log_group_name` – Optional. Specifies the log group name to use in CloudWatch Logs. The maximum length is 512 characters. You can use `{instance_id}`, `{hostname}`, `{local_hostname}`, and `{ip_address}` as variables within the name. If you omit this field, the agent uses `/aws/cwagent/files`. On Kubernetes, the agent includes the cluster name, as in `/aws/cwagent/{cluster_name}/files`.
    - `log_stream_name` – Optional. Specifies the log stream name to use in CloudWatch Logs. The maximum length is 512 characters. This field supports the same variables as `log_group_name`. If you omit this field, the agent uses the host identifier and the log file name, separated by a slash (for example, `i-0123456789abcdef0/app.log`).
    - `multi_line_start_pattern` – Optional. Specifies a regular expression that marks the start of a new log entry. The agent appends lines that don't match the pattern to the previous entry. Specify the value `{timestamp_format}` to derive the pattern from `timestamp_format`.
    - `timestamp_format` – Optional. Specifies the format used to parse timestamps from log lines. For example, `%Y-%m-%d %H:%M:%S`.
    - `timezone` – Optional. Specifies the time zone to use when parsing timestamps. Valid values are UTC and `Local`. The default value is `Local`.
    - `encoding` – Optional. Specifies the encoding of the log files. If you omit this field, the agent uses the default `utf-8`.

- `windows_events` – Optional. Specifies that the agent collects events from the Windows Event Log. This source is available only on servers running Windows Server. It contains one field, `collect_list`.

- `collect_list` – Required if `windows_events` is included. Contains an array of entries, each of which specifies one Windows Event Log channel to collect. The array must contain at least one entry. Each entry can include the following fields:
  - `event_name` – Required. Specifies the Windows Event Log channel to collect from, such as `System` or `Application`. The maximum length is 255 characters.
  - `event_levels` – Optional. Specifies the event severity levels to collect. Valid values are `CRITICAL`, `ERROR`, `WARNING`, `INFORMATION`, and `VERBOSE`. If you omit this field, the agent collects events of all levels.
  - `event_ids` – Optional. Specifies a list of specific event IDs to collect. Each ID must be between 0 and 65535.
  - `event_format` – Optional. Specifies the output format. Valid values are `xml`, which sends the raw XML of the event, and `text`, which sends a structured representation. If you omit this field, the agent uses `text`.
  - `log_group_name` – Optional. Specifies the log group name to use in CloudWatch Logs. The maximum length is 512 characters. If you omit this field, the agent uses `/aws/cwagent/windows_events`.
  - `log_stream_name` – Optional. Specifies the log stream name to use in CloudWatch Logs. The maximum length is 512 characters. If you omit this field, the agent uses the host identifier and the event channel name, separated by a slash.
- `host_metrics` – Optional. Specifies that the agent collects host-level system metrics. This section can include the following field:
  - `collection_interval` – Optional. Specifies how often the agent collects the metrics, in seconds. If you omit this field, the agent uses the `metrics_collection_interval` value from the `agent` section, or 30 seconds if that is also omitted.
- `prometheus` – Optional. Specifies that the agent scrapes metrics from Prometheus targets. This section can include the following field:
  - `config_path` – Required. Specifies the path to the Prometheus scrape configuration file.

> **Note**
>
> Make sure Transaction Search is enabled before you send traces to the OTLP traces endpoint.

The following is an example of an `opentelemetry` section that collects OTLP telemetry and a log file.

```
"opentelemetry": {
  "collect": {
    "otlp": {
      "grpc_endpoint": "0.0.0.0:4317",
      "http_endpoint": "0.0.0.0:4318",
      "span_metrics_enabled": true
    },
    "files": {
      "collect_list": [
        {
          "file_path": "/var/log/app/*.log",
          "log_group_name": "/app/{instance_id}",
          "log_stream_name": "{hostname}",
          "timestamp_format": "%Y-%m-%d %H:%M:%S",
          "timezone": "UTC"
        }
      ]
    }
  }
}
```

CloudWatch agent configuration file: Complete examples

The following is an example of a complete CloudWatch agent configuration file for a Linux server.

The items listed in the `measurement` sections for the metrics you want to collect can either specify the complete metric name such or just the part of the metric name that will be appended to the type of resource. For example, specifying either `reads` or `diskio_reads` in the `measurement` section of the `diskio` section will cause the `diskio_reads` metric to be collected.

This example includes both ways of specifying metrics in the `measurement` section.

```
    {
      "agent": {
        "metrics_collection_interval": 10,
        "logfile": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
```

```
          "cpu": {
            "resources": [
              "*"
            ],
            "measurement": [
              {"name": "cpu_usage_idle", "rename": "CPU_USAGE_IDLE", "unit":
 "Percent"},
              {"name": "cpu_usage_nice", "unit": "Percent"},
              "cpu_usage_guest"
            ],
            "totalcpu": false,
            "metrics_collection_interval": 10,
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "disk": {
            "resources": [
              "/",
              "/tmp"
            ],
            "measurement": [
              {"name": "free", "rename": "DISK_FREE", "unit": "Gigabytes"},
              "total",
              "used"
            ],
             "ignore_file_system_types": [
              "sysfs", "devtmpfs"
            ],
            "metrics_collection_interval": 60,
            "append_dimensions": {
              "customized_dimension_key_3": "customized_dimension_value_3",
              "customized_dimension_key_4": "customized_dimension_value_4"
            }
          },
          "diskio": {
            "resources": [
              "*"
            ],
            "measurement": [
              "reads",
              "writes",
              "read_time",
```

```
            "write_time",
            "io_time"
          ],
          "metrics_collection_interval": 60
        },
        "swap": {
          "measurement": [
            "swap_used",
            "swap_free",
            "swap_used_percent"
          ]
        },
        "mem": {
          "measurement": [
            "mem_used",
            "mem_cached",
            "mem_total"
          ],
          "metrics_collection_interval": 1
        },
        "net": {
          "resources": [
            "eth0"
          ],
          "measurement": [
            "bytes_sent",
            "bytes_recv",
            "drop_in",
            "drop_out"
          ]
        },
        "netstat": {
          "measurement": [
            "tcp_established",
            "tcp_syn_sent",
            "tcp_close"
          ],
          "metrics_collection_interval": 60
        },
        "processes": {
          "measurement": [
            "running",
            "sleeping",
            "dead"
```

```
            ]
          }
        },
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
 ["d1"],[]],
        "force_flush_interval" : 30
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "log_stream_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/test.log",
                "log_group_name": "test.log",
                "log_stream_name": "test.log",
                "timezone": "Local"
              }
            ]
          }
        },
        "log_stream_name": "my_log_stream_name",
        "force_flush_interval" : 15,
        "metrics_collected": {
           "kubernetes": {
                "enhanced_container_insights": true
      }
    }
  }
}
```

The following is an example of a complete CloudWatch agent configuration file for a server running Windows Server.

```
{
      "agent": {
        "metrics_collection_interval": 60,
        "logfile": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\\amazon-
cloudwatch-agent.log"
      },
      "metrics": {
        "namespace": "MyCustomNamespace",
        "metrics_collected": {
          "Processor": {
            "measurement": [
              {"name": "% Idle Time", "rename": "CPU_IDLE", "unit": "Percent"},
              "% Interrupt Time",
              "% User Time",
              "% Processor Time"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_1": "customized_dimension_value_1",
              "customized_dimension_key_2": "customized_dimension_value_2"
            }
          },
          "LogicalDisk": {
            "measurement": [
              {"name": "% Idle Time", "unit": "Percent"},
              {"name": "% Disk Read Time", "rename": "DISK_READ"},
              "% Disk Write Time"
            ],
            "resources": [
              "*"
            ]
          },
          "customizedObjectName": {
            "metrics_collection_interval": 60,
            "customizedCounterName": [
              "metric1",
              "metric2"
            ],
```

```
            "resources": [
              "customizedInstances"
            ]
          },
          "Memory": {
            "metrics_collection_interval": 5,
            "measurement": [
              "Available Bytes",
              "Cache Faults/sec",
              "Page Faults/sec",
              "Pages/sec"
            ]
          },
          "Network Interface": {
            "metrics_collection_interval": 5,
            "measurement": [
              "Bytes Received/sec",
              "Bytes Sent/sec",
              "Packets Received/sec",
              "Packets Sent/sec"
            ],
            "resources": [
              "*"
            ],
            "append_dimensions": {
              "customized_dimension_key_3": "customized_dimension_value_3"
            }
          },
          "System": {
            "measurement": [
              "Context Switches/sec",
              "System Calls/sec",
              "Processor Queue Length"
            ]
          }
        },
        "append_dimensions": {
          "ImageId": "${aws:ImageId}",
          "InstanceId": "${aws:InstanceId}",
          "InstanceType": "${aws:InstanceType}",
          "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
        },
        "aggregation_dimensions" : [["ImageId"], ["InstanceId", "InstanceType"],
["d1"],[]]
```

```
      },
      "logs": {
        "logs_collected": {
          "files": {
            "collect_list": [
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\amazon-cloudwatch-agent.log",
                "log_group_name": "amazon-cloudwatch-agent.log",
                "timezone": "UTC"
              },
              {
                "file_path": "c:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\Logs\
\test.log",
                "log_group_name": "test.log",
                "timezone": "Local"
              }
            ]
          },
          "windows_events": {
            "collect_list": [
              {
                "event_name": "System",
                "event_levels": [
                  "INFORMATION",
                  "ERROR"
                ],
                "log_group_name": "System",
                "log_stream_name": "System",
                "event_format": "xml"
              },
              {
                "event_name": "CustomizedName",
                "event_levels": [
                  "WARNING",
                  "ERROR"
                ],
                "log_group_name": "CustomizedLogGroup",
                "log_stream_name": "CustomizedLogStream",
                "event_format": "xml"
              }
            ]
          }
        },
```

```
        "log_stream_name": "example_log_stream_name"
      }
    }
```

Save the CloudWatch agent configuration file manually

If you create or edit the CloudWatch agent configuration file manually, you can give it any name. After you have created the file, you can copy it to other servers where you want to run the agent.

Update the CloudWatch agent configuration file manually

Creating or naming a configuration file does not apply it to the agent. When you apply a configuration, the agent stores it and generates the files that it uses at runtime. On a Linux server, these files are in the `/opt/aws/amazon-cloudwatch-agent/etc` directory, as shown in the following structure.

```
/opt/aws/amazon-cloudwatch-agent/etc/
    cloudwatch-agent.json         (your source configuration file; you can give it any
 name)
    amazon-cloudwatch-agent.d/    (agent-managed directory of active configuration
 fragments)
        file_name              (a fragment copied from a local file source)
        ssm_name               (a fragment copied from a Systems Manager Parameter
 Store source)
    amazon-cloudwatch-agent.toml  (generated file that the systemd or upstart service
 uses to run the agent)
```

The `amazon-cloudwatch-agent.d` directory holds the active configuration that the agent manages. Keep your source configuration file in a location that you choose, and do not use the `amazon-cloudwatch-agent.d` directory to store source files.

To apply or update the configuration, use the following commands. For the full command syntax and the behavior when configuration file names match, see Creating multiple CloudWatch agent configuration files.

- To apply a source configuration file, run `amazon-cloudwatch-agent-ctl -a fetch-config -c file:`*`configuration-file-path`*. This command applies the initial configuration or replaces the active configuration set.
- To add another configuration to a running agent without replacing the current one, use the `append-config` option. The `append-config` option preserves the existing configuration fragments, unless a configuration file name matches one that the agent is already using.

> **Note**
>
> The systemd and upstart services use the generated `amazon-cloudwatch-agent.toml` file to run the agent. If this file is absent, starting the agent can apply the default configuration. The default configuration can replace the active configuration in the `amazon-cloudwatch-agent.d` directory. Apply your configuration with `fetch-config` or `append-config` before you start the agent.

Uploading the CloudWatch agent configuration file to Systems Manager Parameter Store

If you plan to use the SSM Agent to install the CloudWatch agent on servers, after you manually edit the CloudWatch agent configuration file, you can upload it to Systems Manager Parameter Store. To do so, use the Systems Manager `put-parameter` command.

To be able to store the file in Parameter Store, you must use an IAM role with sufficient permissions.

Use the following command, where *`parameter name`* is the name to be used for this file in Parameter Store and *`configuration_file_pathname`* is the path and file name of the configuration file that you edited.

```
aws ssm put-parameter --name "parameter name" --type "String" --value
 file://configuration_file_pathname
```

Enable CloudWatch Application Signals

Use CloudWatch Application Signals to automatically instrument your applications on AWS so that you can track application performance against your business objectives. Application Signals provides you a unified, application-centric view of your Java applications, their dependencies, and their edges. For more information, see Application Signals.

CloudWatch Application Signals leverages the CloudWatch agent to receive metrics and traces from your auto-instrumented applications, optionally apply rules to reduce high cardinality, and then publish the processed telemetry to CloudWatch. You can provide custom configuration to the CloudWatch agent specifically for Application Signals using the agent configuration file. To start with, the presence of an `application_signals` section under the `metrics_collected` section within the `logs` section of the agent configuration file specifies that the CloudWatch

agent will receive metrics from your auto-instrumented applications. Similarly, the presence of an `application_signals` section under the `traces_collected` section within the `traces` section of the agent configuration file specifies that the CloudWatch agent is enabled to receive traces from your auto-instrumented applications. In addition, you can optionally pass in custom configuration rules to reduce publishing high-cardinality telemetry as outlined in this section.

- For Amazon EKS clusters, when you install the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent is by default enabled to receive both metrics and traces from your auto-instrumented applications. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration to the Amazon EKS add-on when you create or update it by using additional configuration, as outlined in (Optional) Additional configuration.
- For RedHat for OpenShift on AWS (ROSA), when you install the CloudWatch agent operator using helm charts, the CloudWatch agent is by default enabled to receive both metrics and traces from your auto-instrumented applications. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration by using the Helm chart, as outlined in (Optional) Additional configuration.
- For other supported platforms including Amazon EC2, you must start the CloudWatch agent with an agent configuration that enables Application Signals by specifying the `application_signals` sections and optionally any custom configuration rules as outlined later in this section.

The following is an overview of the fields in the CloudWatch agent configuration file that are related to CloudWatch Application Signals.

- `logs`
  - `metrics_collected` – This field can contain sections to specify that the agent is to collect logs to enable use cases such as CloudWatch Application Signals and Container Insights with enhanced observability for Amazon EKS.

> **Note**
>
> Previously this section was also used to specify that the agent is to collect logs that are in embedded metric format. Those settings are no longer needed.

- `application_signals` (Optional) Specifies that you want to enable CloudWatch Application Signals to receive metrics from your auto-instrumented applications to facilitate CloudWatch Application Signals.
  - `rules` (Optional) An array of rules to conditionally select metrics and traces and apply actions to handle high-cardinality scenarios. Each rule can contain the following fields:
    - `rule_name` (Optional) The name of the rule.
    - `selectors` (Optional) An array of metrics and traces dimension matchers. Each selector must provide the following fields:
      - `dimension` Required if `selectors` is not empty. This specifies the dimension of metrics and traces to use as a filter.
      - `match` Required if `selectors` is not empty. A wildcard pattern used for matching values of the specified dimension.
    - `action` (Optional) The action to be applied to metrics and traces that match the specified selectors. The value of `action` must be one of the following keywords:
      - `keep` Specifies to send only the metrics and traces to CloudWatch if matched by the `selectors`.
      - `drop` Specifies to drop the metric and traces that match the `selectors`.
      - `replace` Specifies to replace the dimensions of the metrics and traces that match `selectors`. They are replaced according to the `replacements` section.
    - `replacements` Required if `action` is `replace`. An array of dimension and value pairs that will be applied to metrics and traces that match the specified `selectors` when the `action` is `replace`. Each replacement must provide the following fields:
      - `target_dimension` Required if `replacements` is not empty. Specifies the dimension that needs to be replace.
      - `value` Required if `replacements` is not empty. The value to replace the original value of `target_dimension` with.
  - `limiter` (Optional) Use this section to limit how many metrics and dimensions Application Signals sends to CloudWatch, to optimize your costs.
    - `disabled` (Optional) If `true`, the metric limiting feature is disabled. The default is `false`
    - `drop_threshold` (Optional) The maximum number of distinct metrics per service in one rotation interval that can be exported by one CloudWatch agent. The default is 500.

- `rotation_interval` (Optional) The interval at which the limiter resets the metric records for distinction counting. This is expressed as a string with a sequence of numbers and a unit suffix. Fractions are supported. The supported unit suffixes are `s`, `m`, `h`, `ms`, `us`, and `ns`. The default is `1h` for one hour.
- `log_dropped_metrics` (Optional) Specifies whether the agent should write logs to the CloudWatch agent logs when Application Signals metrics are dropped. The default is `false`.

> **Note**
>
> To activate this logging, the `debug` parameter in the `agent` section must also be set to `true`.

- `traces`
  - `traces_collected`
    - `application_signals` Optional. Specify this to enable the CloudWatch agent to receive traces from your auto-instrumented applications for facilitating CloudWatch Application Signals.

> **Note**
>
> Even though the custom `application_signals` rules are specified under the `metrics_collected` section that is contained in the `logs` section, they also implicitly apply to the `traces_collected` section as well. The same set of rules will apply to both metrics and traces.

When there are multiple rules with different actions, they apply in the following sequence: `keep`, then `drop`, then `replace`.

The following is an example of a full CloudWatch agent configuration file that applies custom rules.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
```

```
        "rules": [
          {
            "rule_name": "keep01",
            "selectors": [
              {
                "dimension": "Service",
                "match": "pet-clinic-frontend"
              },
              {
                "dimension": "RemoteService",
                "match": "customers-service"
              }
            ],
            "action": "keep"
          },
          {
            "rule_name": "drop01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "GET /api/customer/owners/*"
              }
            ],
            "action": "drop"
          },
          {
            "rule_name": "replace01",
            "selectors": [
              {
                "dimension": "Operation",
                "match": "PUT /api/customer/owners/*/pets/*"
              },
              {
                "dimension": "RemoteOperation",
                "match": "PUT /owners"
              }
            ],
            "replacements": [
              {
                "target_dimension": "Operation",
                "value": "PUT /api/customer/owners/{ownerId}/pets{petId}"
              }
            ],
            "action": "replace"
```

```
          }
        ]
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}
```

For the previous example configuration file, the `rules` are processed as follows:

1. Rule `keep01` ensures that any metrics and traces with the dimension `Service` as `pet-clinic-frontend` and the dimension `RemoteService` as `customers-service` are kept.
2. For the processed metrics and traces after applying `keep01`, the `drop01` rule ensures that metrics and traces with the dimension `Operation` as `GET /api/customer/owners/*` are dropped.
3. For the processed metrics and traces after applying `drop01`, the `replace01` rule updates metrics and traces that have the dimension `Operation` as `PUT /api/customer/owners/*/pets/*` and the dimension `RemoteOperation` as `PUT /owners` such that their `Operation` dimension is now replaced to be `PUT /api/customer/owners/{ownerId}/pets{petId}`.

The following is a complete example of a CloudWatch configuration file that manages cardinality in Application Signals by changing the metric limit to 100, enabling the logging of dropped metrics, and setting the rotation interval to two hours.

```
{
    "logs": {
        "metrics_collected": {
            "application_signals": {
                "limiter": {
                    "disabled": false,
                    "drop_threshold": 100,
                    "rotation_interval": "2h",
                    "log_dropped_metrics": true
                }
            }
        },
```

```
        "traces": {
            "traces_collected": {
                "application_signals": {}
            }
        }
    }
}
```

Collect network performance metrics

EC2 instances running on Linux that use the Elastic Network Adapter (ENA) publish network performance metrics. Version 1.246396.0 and later of the CloudWatch agent enable you to import these network performance metrics into CloudWatch. When you import these network performance metrics into CloudWatch, they are charged as CloudWatch custom metrics.

For more information about the ENA driver, see Enabling enhanced networking with the Elastic Network Adapter (ENA) on Linux instances and Enabling enhanced networking with the Elastic Network Adapter (ENA) on Windows instances.

How you set up the collection of network performance metrics differs on Linux servers and Windows servers.

The following table lists these network performance metrics enabled by the ENA adapter. When the CloudWatch agent imports these metrics into CloudWatch from Linux instances, it prepends `ethtool_` at the beginning of each of these metric names.

Metric	Description
Name on Linux servers: **`bw_in_allowance_exceeded`** Name on Windows servers: **`Aggregate inbound BW allowance exceeded`**	The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
Name on Linux servers: **bw_out_allowance_exceeded** Name on Windows servers: **Aggregate outbound BW allowance exceeded**	The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
Name on Linux servers:**conntrack_allowance_available** Name on Windows servers: **Available connection tracking allowance**	Reports the number of tracked connections that can be established by the instance before hitting the Connections Tracked allowance of that instance type. This metric is available only on Nitro-based EC2 instances using the Linux driver for Elastic Network Adapter (ENA) starting from version 2.8.1, and on computers using the Windows driver for Elastic Network Adapter (ENA) starting from version 2.6.0. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
Name on Linux servers:**ena_srd_mode** Name on Windows servers: **ena srd mode**	Describes which ENA Express features are enabled. For more information about ENA Express, see Improve network performance with ENA Express on Linux instances Values are as follows: • 0 = ENA Express off, UDP off • 1 = ENA Express on, UDP off • 2 = ENA Express off, UDP on **Note** This happens only when ENA Express was originally enabled, and UDP was configure d to use it. The prior value is retained for UDP traffic. • 3 = ENA Express on, UDP on
Name on Linux servers:**ena_srd_e ligible_tx_pkts** Name on Windows servers: **ena srd eligible tx pkts**	The number of network packets sent within a given time period that meet AWS Scalable Reliable Datagram (SRD) requirements for eligibility, as follows: • Both sending and receiving instance types are supported. • Both sending and receiving instances must have ENA Express configured. • The sending and receiving instances must be on the same subnet. • The network path between the instances must not include middleware boxes. ENA Express doesn't currently support middleware boxes.

Metric	Description
Name on Linux servers:**ena_srd_tx_pkts** Name on Windows servers: **ena srd tx pkts**	The number of SRD packets transmitted within a given time period.
Name on Linux servers:**ena_srd_rx_pkts** Name on Windows servers: **ena srd rx pkts**	The number of SRD packets received within a given time period.
Name on Linux servers:**ena_srd_resource_utilization** Name on Windows servers: **ena srd resource utilization**	The percentage of the maximum allowed memory utilization for concurrent SRD connections that the instance has consumed.
Name on Linux servers: **linklocal_allowance_exceeded** Name on Windows servers: **Link local packet rate allowance exceeded**	The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service, but does not impact traffic to custom DNS resolvers. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
Name on Linux servers: **pps_allowance_exceeded** Name on Windows servers: **PPS allowance exceeded**	The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance. This metric is collected only if you have listed it in the `ethtool` subsection of the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Linux setup

On Linux servers, the *ethtool plugin* enables you to import the network performance metrics into CloudWatch.

ethtool is a standard Linux utility that can collect statistics about Ethernet devices on Linux servers. The statistics it collects depend on the network device and driver. Examples of these statistics include `tx_cnt`, `rx_bytes`, `tx_errors`, and `align_errors`. When you use the ethtool plugin with the CloudWatch agent, you can also import these statistics into CloudWatch, along with the EC2 network performance metrics listed earlier in this section.

> **Tip**
>
> To find the statistics available on our operating system and network device, use the `ethtool -S` command.

When the CloudWatch agent imports metrics into CloudWatch, it adds an `ethtool_` prefix to the names of all imported metrics. So the standard ethtool statistic `rx_bytes` is called `ethtool_rx_bytes` in CloudWatch, and the EC2 network performance metric `bw_in_allowance_exceeded` is called `ethtool_bw_in_allowance_exceeded` in CloudWatch.

On Linux servers, to import ethtool metrics, add an `ethtool` section to the `metrics_collected` section of the CloudWatch agent configuration file. The `ethtool` section can include the following subsections:

- **interface_include**— Including this section causes the agent to collect metrics from only the interfaces that have names listed in this section. If you omit this section, metrics are collected from all Ethernet interfaces that aren't listed in `interface_exclude`.

  The default ethernet interface is `eth0`.
- **interface_exclude**— If you include this section, list the Ethernet interfaces that you don't want to collect metrics from.

  The ethtool plugin always ignores loopback interfaces.
- **metrics_include**— This section lists the metrics to import into CloudWatch. It can include both standard statistics collected by ethtool and Amazon EC2 high-resolution network metrics.

The following example displays part of the CloudWatch agent configuration file. This configuration collects the standard ethtool metrics `rx_packets` and `tx_packets`, and the Amazon EC2 network performance metrics from only the `eth1` interface.

For more information about the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

```
{
"metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "ethtool": {
        "interface_include": [
          "eth1"
        ],
        "metrics_include": [
          "bw_in_allowance_exceeded",
          "bw_out_allowance_exceeded",
          "conntrack_allowance_exceeded",
          "linklocal_allowance_exceeded",
          "pps_allowance_exceeded"
         ]
```

```
      }
   }
}
}
```

Windows setup

On Windows servers, the network performance metrics are available through Windows Performance Counters, which the CloudWatch agent already collects metrics from. So you do not need a plugin to collect these metrics from Windows servers.

The following is a sample configuration file to collect network performance metrics from Windows. For more information about editing the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

```
{
    "metrics": {
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "ENA Packets Shaping": {
                "measurement": [
                    "Aggregate inbound BW allowance exceeded",
                    "Aggregate outbound BW allowance exceeded",
                    "Connection tracking allowance exceeded",
                    "Link local packet rate allowance exceeded",
                    "PPS allowance exceeded"
                ],
                "metrics_collection_interval": 60,
                "resources": [
                    "*"
                ]
            }
        }
    }
}
```

Viewing network performance metrics

After importing network performance metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a

threshold that you specify. The following procedure shows how to view ethtool metrics as a time series graph. For more information about setting alarms, see Using Amazon CloudWatch alarms.

Because all of these metrics are aggregate counters, you can use CloudWatch metric math functions such as `RATE(METRICS())` to calculate the rate for these metrics in graphs or use them to set alarms. For more information about metric math functions, see Math expressions with metrics.

To view network performance metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. You can do the following:
    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
    b. To sort the table, use the column heading.
    c. To filter by resource, choose the resource ID, and then choose **Add to search**.
    d. To filter by metric, choose the metric name, and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, and then choose **Add to dashboard**.

Collect Amazon EBS NVMe driver metrics

For CloudWatch agent to collect AWS NVMe driver metrics for Amazon EBS volumes attached to an Amazon EC2 instance, add the `diskio` section inside the `metrics_collected` section of the CloudWatch agent configuration file.

Additionally, the CloudWatch agent binary requires `ioctl` permissions for NVMe driver devices to collect metrics from attached Amazon EBS volumes.

The following metrics can be collected.

Metric	Metric name in CloudWatch	Description
ebs_total_read_ops	diskio_ebs_total_read_ops	The total number of completed read operations.
ebs_total_write_ops	diskio_ebs_total_write_ops	The total number of completed write operations.
ebs_total_read_bytes	diskio_ebs_total_read_bytes	The total number of read bytes transferred.
ebs_total_write_bytes	diskio_ebs_total_write_bytes	The total number of write bytes transferred.
ebs_total_read_time	diskio_ebs_total_read_time	The total time spent, in microseconds, by all completed read operations.
ebs_total_write_time	diskio_ebs_total_write_time	The total time spent, in microseconds, by all completed write operations.
ebs_volume_performance_exceeded_iops	diskio_ebs_volume_performance_exceeded_iops	The total time, in microseconds, that IOPS demand exceeded the volume's provisioned IOPS performance.
ebs_volume_performance_exceeded_tp	diskio_ebs_volume_performance_exceeded_tp	The total time, in microseconds, that throughput demand exceeded the volume's provisioned throughput performance.
ebs_ec2_instance_performanc	diskio_ebs_ec2_instance_performance_exceeded_iops	The total time, in microseconds, that the EBS volume exceeded the attached Amazon EC2 instance's maximum IOPS performance.

Metric	Metric name in CloudWatch	Description
e_exceeded_iops		
ebs_ec2_instance_performance_exceeded_tp	diskio_ebs_ec2_instance_performance_exceeded_tp	The total time, in microseconds, that the EBS volume exceeded the attached Amazon EC2 instance's maximum throughput performance.
ebs_volume_queue_length	diskio_ebs_volume_queue_length	The number of read and write operations waiting to be completed.

Collect Amazon EC2 instance store volume NVMe driver metrics

For CloudWatch agent to collect AWS NVMe driver metrics for instance store volumes attached to an Amazon EC2 instance, add the `diskio` section inside the `metrics_collected` section of the CloudWatch agent configuration file.

Additionally, the CloudWatch agent binary requires `ioctl` permissions for NVMe driver devices to collect metrics from attached instance store volumes.

The following metrics can be collected.

Metric	Metric name in CloudWatch	Description
instance_store_total_read_ops	diskio_instance_store_total_read_ops	The total number of completed read operations.
instance_store_total_write_ops	diskio_instance_store_total_write_ops	The total number of completed write operations.

Metric	Metric name in CloudWatch	Description
instance_store_total_read_bytes	diskio_instance_store_total_read_bytes	The total number of read bytes transferred.
instance_store_total_write_bytes	diskio_instance_store_total_write_bytes	The total number of write bytes transferred.
instance_store_total_read_time	diskio_instance_store_total_read_time	The total time spent, in microseconds, by all completed read operations.
instance_store_total_write_time	diskio_instance_store_total_write_time	The total time spent, in microseconds, by all completed write operations.
instance_store_performance_exceeded_iops	diskio_instance_store_performance_exceeded_iops	The total time, in microseconds, that IOPS demand exceeded the volume's IOPS maximum performance.
instance_store_performance_exceeded_tp	diskio_instance_store_performance_exceeded_tp	The total time, in microseconds, that throughput demand exceeded the volume's maximum throughput performance.
instance_store_volume_queue_length	diskio_instance_store_volume_queue_length	The number of read and write operations waiting to be completed.

Collect NVIDIA GPU metrics

You can use the CloudWatch agent to collect NVIDIA GPU metrics from Linux servers. To set this up, add a `nvidia_gpu` section inside the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Linux section.

Additionally, the instance must have an NVIDIA driver installed. NVIDIA drivers on pre-installed on some Amazon Machine Images (AMIs). Otherwise, you can manually install the driver. For more information, see Install NVIDIA drivers on Linux instances.

The following metrics can be collected. All of these metrics are collected with no CloudWatch `Unit`, but you can specify a unit for each metric by adding a parameter to the CloudWatch agent configuration file. For more information, see Linux section.

Metric	Metric name in CloudWatch	Description
`utilization_gpu`	`nvidia_smi_utilization_gpu`	The percentage of time over the past sample period during which one or more kernels on the GPU was running.
`temperature_gpu`	`nvidia_smi_temperature_gpu`	The core GPU temperature in degrees Celsius.
`power_draw`	`nvidia_smi_power_draw`	The last measured power draw for the entire board, in watts.
`utilization_memory`	`nvidia_smi_utilization_memory`	The percentage of time over the past sample period during which global (device) memory was being read or written.
`fan_speed`	`nvidia_smi_fan_speed`	The percentage of maximum fan speed that the device's fan is currently intended to run at.
`memory_total`	`nvidia_smi_memory_total`	Reported total memory, in MB.
`memory_used`	`nvidia_smi_memory_used`	Memory used, in MB.

Metric	Metric name in CloudWatch	Description
memory_free	nvidia_smi_memory_free	Memory free, in MB.
pcie_link_gen_current	nvidia_smi_pcie_link_gen_current	The current link generation.
pcie_link_width_current	nvidia_smi_pcie_link_width_current	The current link width.
encoder_stats_session_count	nvidia_smi_encoder_stats_session_count	Current number of encoder sessions.
encoder_stats_average_fps	nvidia_smi_encoder_stats_average_fps	The moving average of the encode frames per second.
encoder_stats_average_latency	nvidia_smi_encoder_stats_average_latency	The moving average of the encode latency in microseconds.
clocks_current_graphics	nvidia_smi_clocks_current_graphics	The current frequency of the graphics (shader) clock.
clocks_current_sm	nvidia_smi_clocks_current_sm	The current frequency of the Streaming Multiprocessor (SM) clock.
clocks_current_memory	nvidia_smi_clocks_current_memory	The current frequency of the memory clock.

Metric	Metric name in CloudWatch	Description
clocks_current_video	nvidia_smi_clocks_current_video	The current frequency of the video (encoder plus decoder) clocks.

All of these metrics are collected with the following dimensions:

Dimension	Description
index	A unique identifier for the GPU on this server. Represents the NVIDIA Management Library (NVML) index of the device.
name	The type of GPU. For example, NVIDIA Tesla A100
arch	The server architecture.

Collect EFA metrics

You can use the CloudWatch agent to collect Elastic Fabric Adapter (EFA) metrics from Linux servers that have EFA devices attached. To set this up, add an `efa` section inside the `metrics_collected` section of the CloudWatch agent configuration file. For more information, see Linux section.

> **Note**
>
> All EFA metrics are reported as delta values. The agent automatically applies cumulative-to-delta conversion.

The following metrics can be collected.

Metric	Description
`efa_tx_bytes`	Bytes transmitted on the EFA device.
`efa_rx_bytes`	Bytes received on the EFA device.
`efa_tx_pkts`	Packets transmitted on the EFA device.
`efa_rx_pkts`	Packets received on the EFA device.
`efa_rx_dropped`	Received packets dropped on the EFA device.
`efa_send_bytes`	Bytes sent through EFA send operations.
`efa_send_wrs`	Send work requests completed.
`efa_recv_bytes`	Bytes received through EFA receive operations.
`efa_recv_wrs`	Receive work requests completed.
`efa_rdma_read_bytes`	Bytes transferred through RDMA read operations.
`efa_rdma_read_resp_bytes`	Bytes received in RDMA read responses.
`efa_rdma_read_wrs`	RDMA read work requests completed.
`efa_rdma_read_wr_err`	RDMA read work request errors.
`efa_rdma_write_bytes`	Bytes transferred through RDMA write operations.

Metric	Description
efa_rdma_write_recv_bytes	Bytes received through RDMA write operations.
efa_rdma_write_wrs	RDMA write work requests completed.
efa_rdma_write_wr_err	RDMA write work request errors.
efa_retrans_bytes	Bytes retransmitted.
efa_retrans_pkts	Packets retransmitted.
efa_retrans_timeout_events	Retransmission timeout events.
efa_impaired_remote_conn_events	Impaired remote connection events.
efa_unresponsive_remote_events	Unresponsive remote endpoint events.

Dimensions

Each EFA metric is emitted with the following dimensions by default:

- `device` – The EFA device name.
- `port` – The EFA port number.
- `eniId` – The ENI ID of the EFA NIC.

To append additional dimensions such as `InstanceId`, use the `append_dimensions` field in the `metrics` section of the agent configuration file.

Example configuration

The following example configuration collects a subset of EFA metrics every 60 seconds and appends the `InstanceId` dimension to each metric.

```
{
  "metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "efa": {
        "measurement": [
          "efa_tx_bytes",
          "efa_rx_bytes",
          "efa_tx_pkts",
          "efa_rx_pkts",
          "efa_rx_dropped",
          "efa_rdma_read_bytes",
          "efa_rdma_write_bytes"
        ],
        "metrics_collection_interval": 60
      }
    }
  }
}
```

Collect Java Management Extensions (JMX) metrics

You can use the CloudWatch agent to collect Java Management Extensions (JMX) metrics from your Java applications.

The CloudWatch agent supports collecting these metrics from the following versions:

- JVM 8 and later
- Kafka 0.8.2.x and later
- Tomcat 9, 10.1, and 11 (beta)

Amazon EC2

To enable JMX in your JVM instance

For the CloudWatch agent to be able to collect JMX metrics, your application's JVM must bind to a port using the `com.sun.management.jmxremote.port` system property.

```
java -Dcom.sun.management.jmxremote.port=port-number -jar example.jar
```

For more information and other configurations, see the JMX documentation.

Amazon EKS

To enable JMX on your Java application pods

When using the CloudWatch Observability EKS add-on, add the following annotations to the workload manifest file under the `PodTemplate` section to enable JMX metrics collection:

- `instrumentation.opentelemetry.io/inject-java: "true"`
- One or more of the following:
  - For JVM metrics: `cloudwatch.aws.amazon.com/inject-jmx-jvm: "true"`
  - For Kafka broker metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka: "true"`
  - For Kafka consumer metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka-consumer: "true"`
  - For Kafka producer metrics: `cloudwatch.aws.amazon.com/inject-jmx-kafka-producer: "true"`
  - For Tomcat metrics: `cloudwatch.aws.amazon.com/inject-jmx-tomcat: "true"`

You must also configure a `jmx` section in the CloudWatch agent configuration as described below. For more information about providing a custom agent configuration with the EKS add-on, see Use a custom CloudWatch agent configuration.

Fields common to Amazon EC2 and Amazon EKS

To start collecting JMX metrics, add a `jmx` section inside the `metrics_collected` section of the CloudWatch agent configuration file. The `jmx` section must include at least one of the following target subsections.

- `jvm` – Optional. Specifies that you want to retrieve Java Virtual Machine (JVM) metrics from the instance. For more information, see Collect JVM metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of JVM metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect JVM metrics.

Within the entry for each individual metric, you can optionally specify one or both of the following:

- `rename` – Specifies a different name for this metric.
- `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka` – Optional. Specifies that you want to retrieve Apache Kafka broker metrics from the instance. For more information, see Collect Kafka metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Kafka broker metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the first table in Collect Kafka metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka-consumer` – Optional. Specifies that you want to retrieve Apache Kafka consumer metrics from the instance. For more information, see Collect Kafka metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Kafka consumer metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the second metrics table in Collect Kafka metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `kafka-producer` – Optional. Specifies that you want to retrieve Apache Kafka producer metrics from the instance. For more information, see Collect Kafka metrics.

This section can include the following fields:

- `measurement` – Specifies the array of Kafka producer metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the third metrics table in Collect Kafka metrics.

  Within the entry for each individual metric, you can optionally specify one or both of the following:

  - `rename` – Specifies a different name for this metric.
  - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

- `tomcat` – Optional. Specifies that you want to retrieve Apache Tomcat metrics from the instance. For more information, see Collect Tomcat metrics.

  This section can include the following fields:

  - `measurement` – Specifies the array of Tomcat metrics to be collected. For a list of the possible values to use here, see the **Metric** column in the table in Collect Tomcat metrics.

    Within the entry for each individual metric, you can optionally specify one or both of the following:

    - `rename` – Specifies a different name for this metric.
    - `unit` – Specifies the unit to use for this metric, overriding the default unit for the metric. The unit that you specify must be a valid CloudWatch metric unit, as listed in the `Unit` description in MetricDatum.

The `jmx` section can also include the optional `append_dimensions` field:

- `append_dimensions` – Optional. Additional dimensions to use for only the process metrics. If you specify this field, it's used in addition to dimensions specified in the `append_dimensions` field that is used for all types of metrics collected by the agent.

Fields only on Amazon EC2

The following fields apply only when running the CloudWatch agent on Amazon EC2 instances.

- `endpoint` – The address for the JMX client to connect to. The format is `ip:port`. If the endpoint is not localhost, both password authentication and SSL must be enabled.
- `metrics_collection_interval` – Optional. Specifies how often to collect the processes metrics, overriding the global `metrics_collection_interval` specified in the `agent` section of the configuration file.

  This value is specified in seconds. For example, specifying 10 causes metrics to be collected every 10 seconds, and setting it to 300 specifies metrics to be collected every 5 minutes.

  If you set this value below 60 seconds, each metric is collected as a high-resolution metric. For more information, see High-resolution metrics.

If JMX was enabled with password authentication or SSL for remote access, you can use the following fields.

- `password_file` – Optional. Specifies a Java properties file of keys to passwords. The file must be read-only and restricted to the user running the CloudWatch agent. If password authentication is enabled, this requires the same username and password pair as the entry in the JMX password file provided in the `com.sun.management.jmxremote.password.file` property. If SSL is enabled, it requires entries for `keystore` and `truststore` and corresponds to the `javax.net.ssl.keyStorePassword` and `javax.net.ssl.trustStorePassword` respectively.
- `username` – If password authentication is enabled, specify the username that matches the username in the provided password file.
- `keystore_path` – If SSL is enabled, specify the full path to the Java keystore, which consists of a private key and a certificate to the public key. Corresponds to the `javax.net.ssl.keyStore` property.
- `keystore_type` – If SSL is enabled, specify the type of keystore being used. Corresponds to the `javax.net.ssl.keyStoreType` property.
- `truststore_path` – If SSL is enabled, specify the full path to the Java truststore, which must contain the remote JMX server's public certificate. Corresponds to the `javax.net.ssl.trustStore` property.
- `truststore_type` – If SSL is enabled, specify the type of truststore being used. Corresponds to the `javax.net.ssl.trustStoreType` property.
- `remote_profile` – Optional. Supported JMX remote profiles are TLS in combination with SASL profiles: `SASL/PLAIN`, `SASL/DIGEST-MD5`, and `SASL/CRAM-MD5`. Should be one of:`SASL/`

PLAIN, SASL/DIGEST-MD5, SASL/CRAM-MD5, TLS SASL/PLAIN, TLS SASL/DIGEST-MD5, or TLS SASL/CRAM-MD5

- realm – Optional. The realm as required by the remote profile SASL/DIGEST-MD5.
- registry_ssl_enabled – If RMI registry authentication is enabled. Set to true if the JVM was configured with com.sun.management.jmxremote.registry.ssl=true.
- insecure – Optional. Set to true to opt out of the validation required if the agent is configured for a non-localhost endpoint.

The following is an example of the jmx section of the CloudWatch agent configuration file.

```
{
  "metrics": {
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "remotehost:1314",
          "jvm": {
            "measurement": [
              "jvm.memory.heap.init",
              "jvm.memory.nonheap.used"
            ]
          },
          "kafka": {
            "measurement": [
              "kafka.request.count",
              {
                "name": "kafka.message.count",
                "rename": "KAFKA_MESSAGE_COUNT",
                "unit": "Count"
              }
            ]
          },
          "username": "cwagent",
          "keystore_path": "/path/to/keystore",
          "keystore_type": "PKCS12",
          "truststore_path": "/path/to/truststore",
          "truststore_type": "PKCS12"
        },
        {
          "endpoint": "localhost:1315",
          "kafka-producer": {
```

```
            "measurement": [
              "kafka.producer.request-rate"
            ]
          },
          "append_dimensions": {
            "service.name": "kafka/1"
          }
        }
      ]
    }
  }
}
```

Collect JVM metrics

You can use the CloudWatch agent to collect Java Virtual Machine (JVM) metrics. To set this up, add a `jvm` section inside the `jmx`section of the CloudWatch agent configuration file.

The following metrics can be collected.

Metric	Dimensions	Description
`jvm.classes.loaded`	[DEFAULT]	The total number of loaded classes. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`jvm.gc.collections.count`	[DEFAULT], `name`	The total number of garbage collections that have occurred.

Metric	Dimensions	Description
		Unit: None **Meaningful statistics:** Minimum, Maximum, Average
`jvm.gc.collections.elapsed`	[DEFAULT], `name`	The approximate accumulated garbage collection elapsed time. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
jvm.memory.heap.init	[DEFAULT]	The initial amount of memory that the JVM requests from the operating system for the heap. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.heap.max	[DEFAULT]	The maximum amount of memory that can be used for the heap. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
jvm.memory.heap.used	[DEFAULT]	The current heap memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.heap.committed	[DEFAULT]	The amount of memory that is guaranteed to be available for the heap. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
jvm.memory.nonheap.init	[DEFAULT]	The initial amount of memory that the JVM requests from the operating system for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.nonheap.max	[DEFAULT]	The maximum amount of memory that can be used for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
jvm.memory.nonheap.used	[DEFAULT]	The current non-heap memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.nonheap.committed	[DEFAULT]	The amount of memory that is guaranteed to be available for non-heap purposes. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
jvm.memory.pool.init	[DEFAULT], name	The initial amount of memory that the JVM requests from the operating system for the memory pool. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.pool.max	[DEFAULT], name	The maximum amount of memory that can be used for the memory pool. **Unit:** Bytes **Meaningful statistics:** Maximum

Metric	Dimensions	Description
jvm.memory.pool.used	[DEFAULT], name	The current memory pool memory usage. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
jvm.memory.pool.committed	[DEFAULT], name	The amount of memory that is guaranteed to be available for the memory pool. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`jvm.threads.count`	[DEFAULT]	The current number of threads. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

The JVM metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of Manually create or edit the CloudWatch agent configuration file for more information. On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name`, `k8s.deployment.name`, `k8s.namespace.name`, `k8s.node.name`, `k8s.pod.name`, and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.

Dimension	Description
name	For jvm.gc.collections metrics, the value is the garbage collector name. For jvm.memory.pool metrics, the value is the memory pool name.

Collect Kafka metrics

You can use the CloudWatch agent to collect Apache Kafka metrics. To set this up, add one or more of the following subsections inside the jmx section of the CloudWatch agent configuration file.

- Use a kafka section to collect Kafka broker metrics.
- Use a kafka-consumer section to collect Kafka consumer metrics.
- Use a kafka-producer section to collect Kafka producer metrics.

Kafka broker metrics

The following metrics can be collected for Kafka brokers.

Metric	Dimensions	Description
kafka.message.count	[DEFAULT]	The number of messages received by the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.request.count`	[DEFAULT], `type`	The number of requests received by the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.failed`	[DEFAULT], `type`	The number of requests to the Kafka broker that resulted in a failure. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`kafka.request.time.total`	[DEFAULT], `type`	The total time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
`kafka.request.time.50p`	[DEFAULT], `type`	The 50th percentile time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.request.time.99p	[DEFAULT], type	The 99th percentile time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
kafka.request.time.avg	[DEFAULT], type	The average time that the Kafka broker has taken to service requests. **Unit:** Milliseconds **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.network.io`	[DEFAULT], `state`	The number of bytes received by or sent by the Kafka broker. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`kafka.purgatory.size`	[DEFAULT], `type`	The number of requests waiting in purgatory. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.partition.count	[DEFAULT]	The number of partitions on the Kafka broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.partition.offline	[DEFAULT]	The number of partitions that are offline. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.partition.under_replicated	[DEFAULT]	The number of under-replicated partitions. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.isr.operation.count	[DEFAULT], operation	The number of in-sync replica shrink and expand operations. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.max.lag	[DEFAULT]	The maximum lag in messages between follower and leader replicas. **Unit:** None **Meaningful statistics:** Maximum

Metric	Dimensions	Description
kafka.controller.active.count	[DEFAULT]	The number of active controllers on the broker. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.leader.election.rate	[DEFAULT]	Leader election rate. If this increases, it indicates broker failures. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.unclean.election.rate	[DEFAULT]	Unclean leader election rate. If this increases, it indicates broker failures. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.request.queue	[DEFAULT]	The size of the request queue. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.logs.flush.time.count	[DEFAULT]	The logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
kafka.logs.flush.time.median	[DEFAULT]	The 50th percentile value of the logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
kafka.logs.flush.time.99p	[DEFAULT]	The 99th percentile value of the logs flush count. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average

The Kafka broker metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are

Dimension	Description
	collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of Manually create or edit the CloudWatch agent configuration file for more information. On Amazon EKS by default, k8s related context is also published as dimension s of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`type`	The request type. Possible values are `produce`, `fetch`, `fetchconsumer` , and `fetchfollower` .
`state`	The direction of network traffic. Possible values are `in` and `out`.
`operation`	The operation type for the in-sync replica. Possible values are `shrink` and `expand`.

Kafka consumer metrics

The following metrics can be collected for Kafka consumers.

Metric	Dimensions	Description
`kafka.consumer.fetch-rate`	[DEFAULT], `client-id`	The number of fetch requests

Metric	Dimensions	Description
		for all topics per second. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.records-lag-max`	[DEFAULT], `client-id`	The number of messages that the consumer lags behind the producer. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.total.bytes-consumed-rate`	[DEFAULT], `client-id`	The average number of bytes consumed for all topics per second. **Unit:** Bytes **Meaningful statistics:** Average

Metric	Dimensions	Description
kafka.consumer.total.fetch-size-avg	[DEFAULT], client-id	The number of bytes fetched per request for all topics. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
kafka.consumer.total.records-consumed-rate	[DEFAULT], client-id	The average number of records consumed for all topics per second. **Unit:** None **Meaningful statistics:** Average
kafka.consumer.bytes-consumed-rate	[DEFAULT], client-id , topic	The average number of bytes consumed per second. **Unit:** Bytes **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.consumer.fetch-size-avg`	[DEFAULT], `client-id` , `topic`	The number of bytes fetched per request. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`kafka.consumer.records-consumed-rate`	[DEFAULT], `client-id` , `topic`	The average number of records consumed per second. **Unit:** None **Meaningful statistics:** Average

The Kafka consumer metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of Manually create or edit the CloudWatch agent configuration file for more information.

Dimension	Description
	On Amazon EKS by default, k8s related context is also published as dimension s of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`client-id`	The ID of the client.
`topic`	The Kafka topic.

Kafka producer metrics

The following metrics can be collected for Kafka producers.

Metric	Dimensions	Description
`kafka.producer.io-wait-time-ns-avg`	[DEFAULT], `client-id`	The average length of time the I/O thread spent waiting for a socket ready for reads or writes. **Unit:** None **Meaningful statistics:** Average
`kafka.producer.out`	[DEFAULT], `client-id`	The average number of

Metric	Dimensions	Description
`going-byte-rate`		outgoing bytes sent per second to all servers. **Unit:** Bytes **Meaningful statistics:** Average
`kafka.producer.request-latency-avg`	[DEFAULT], `client-id`	The average request latency. **Unit:** Milliseconds **Meaningful statistics:** Average
`kafka.producer.request-rate`	[DEFAULT], `client-id`	The average number of requests sent per second. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
kafka.producer.response-rate	[DEFAULT], client-id	The number of responses received per second. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
kafka.producer.byte-rate	[DEFAULT], client-id , topic	The average number of bytes sent per second for a topic. **Unit:** Bytes **Meaningful statistics:** Average
kafka.producer.compression-rate	[DEFAULT], client-id , topic	The average compression rate of record batches for a topic. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.producer.record-error-rate`	[DEFAULT], `client-id` , `topic`	The average per-second number of record sends that resulted in errors for a topic. **Unit:** None **Meaningful statistics:** Average
`kafka.producer.record-retry-rate`	[DEFAULT], `client-id` , `topic`	The average per-second number of retried record sends for a topic. **Unit:** None **Meaningful statistics:** Average

Metric	Dimensions	Description
`kafka.producer.record-send-rate`	[DEFAULT], `client-id` , `topic`	The average number of records sent per second for a topic. **Unit:** None **Meaningful statistics:** Average

Kafka producer metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of Manually create or edit the CloudWatch agent configuration file for more information. On Amazon EKS by default, k8s related context is also published as dimensions of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.

Dimension	Description
client-id	The ID of the client.
topic	The Kafka topic.

Collect Tomcat metrics

You can use the CloudWatch agent to collect Apache Tomcat metrics. To set this up, add a `tomcat` section inside the `jmx` section of the CloudWatch agent configuration file.

The following metrics can be collected.

Metric	Dimensions	Description
tomcat.sessions	[DEFAULT]	The number of active sessions. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average
tomcat.errors	[DEFAULT], proto_handler	The number of errors encountered. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`tomcat.processing_time`	[DEFAULT], `proto_handler`	The total processing time. **Unit:** Milliseconds **Meaningful statistics:** Minimum, Maximum, Average
`tomcat.traffic`	[DEFAULT], `proto_handler`	The number of bytes received and sent. **Unit:** Bytes **Meaningful statistics:** Minimum, Maximum, Average
`tomcat.threads`	[DEFAULT], `proto_handler`	The number of threads. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Metric	Dimensions	Description
`tomcat.max_time`	[DEFAULT], `proto_handler` , `direction`	Maximum time to process a request. **Unit:** Milliseconds **Meaningful statistics:** Maximum
`tomcat.request_count`	[DEFAULT], `proto_handler`	The total requests. **Unit:** None **Meaningful statistics:** Minimum, Maximum, Average

Tomcat metrics are collected with the following dimensions:

Dimension	Description
[DEFAULT]	On Amazon EC2 by default, the host is also published as a dimension of metrics that are collected by the CloudWatch agent, unless you are using the `append_dimensions` field in the `metrics` section. See `omit_hostname` in the agent section of Manually create or edit the CloudWatch agent configuration file for more information.

Dimension	Description
	On Amazon EKS by default, k8s related context is also published as dimension s of metrics (`k8s.container.name` , `k8s.deployment.name` , `k8s.namespace.name` , `k8s.node.name` , `k8s.pod.name` , and `k8s.replicaset.name`). These can be filtered down using the `aggregation_dimensions` field.
`proto_handler`	The `proto_handler` is an identifier for a connector, which is provided in the `<protocol>-<type>-<port>` format (for example, `http-nio-8080`).
`direction`	The traffic direction. Possible values are `received` and `sent`.

Collect metrics and traces with OpenTelemetry

You can collect metrics and traces from your applications or services using the CloudWatch agent with the OpenTelemetry Protocol (OTLP), which is a popular open source solution. You can use any OpenTelemetry SDK to send metrics and traces to the CloudWatch agent. For more information about the available OpenTelemetry SDKs, see OpenTelemetry Supported Language APIs & SDKs..

To collect OpenTelemetry metrics and traces, add an `otlp` section to the CloudWatch agent configuration file. The section has the following fields:

- `grpc_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry metrics or traces sent using gRPC Remote Procedure Calls. The format is `ip:port`. This address must match the address set for the gRPC exporter in the OpenTelemetry SDK. If you omit this field, the default of `127.0.0.1:4317` is used.
- `http_endpoint` – Optional. Specifies the address for the CloudWatch agent to use to listen for OpenTelemetry metrics or traces sent over HTTP. The format is `ip:port`. This address must match the address set for the HTTP exporter in the OpenTelemetry SDK. If you omit this field, the default of `127.0.0.1:4318` is used.

- `tls` – Optional. Specifies that the server should be configured with TLS.
  - `cert_file` – Path to the TLS certificate to use for TLS required connections.
  - `key_file` – Path to the TLS key to use for TLS required connections.

The `otlp` section can be placed in multiple sections within the CloudWatch agent configuration file depending on how and where you want to send the metrics and traces.

Important

Each `otlp` section requires a unique endpoint and port. For detailed information about splitting the metrics and traces endpoints, see OTLP Exporter Configuration in the OpenTelemetry SDK documentation.

To send metrics to CloudWatch or Amazon Managed Service for Prometheus, add the `otlp` section under `metrics_collected` within the `metrics` section. For more information about sending metrics to different destinations, see Manually create or edit the CloudWatch agent configuration file. The following example shows a configuration that sends metrics to CloudWatch:

Note

If you are running the agent in containerized environments and sending telemetry from outside the agent container's network, make sure to specify the endpoint as `0.0.0.0` instead of the default endpoint `127.0.0.1`.

```
{
  "metrics": {
    "metrics_collected": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
}
```

To send metrics to Amazon CloudWatch Logs using the Embedded metric format (EMF), add the `otlp` section under `metrics_collected` within the `logs` section. This sends the EMF logs by default to the `/aws/cwagent` log group and a generated log stream. The metrics are extracted into the `CWAgent` namespace by default. The following example shows a configuration that sends metrics as EMF logs to CloudWatch Logs:

```
{
  "logs": {
    "metrics_collected": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
}
```

To send traces to AWS X-Ray, add the `otlp` section under `traces_collected` within the `traces` section. The following example shows a configuration that sends traces to X-Ray:

```
{
  "traces": {
    "traces_collected": {
      "otlp": {
        "grpc_endpoint": "127.0.0.1:4317",
        "http_endpoint": "127.0.0.1:4318"
      }
    }
  }
}
```

Collect process metrics with the procstat plugin

The *procstat* plugin enables you to collect metrics from individual processes. The plugin is supported on Linux servers and on servers running supported version of Windows Server. This section describes how to configure the CloudWatch agent for procstat and view metrics the CloudWatch agent imports. It also lists the metrics that procstat collects.

> **Note**
>
> The `procstat` plugin is not supported for the Fargate launch type in Amazon ECS environments.

Topics

- Configure the CloudWatch agent for procstat
- Metrics collected by procstat
- Viewing process metrics imported by the CloudWatch agent

Configure the CloudWatch agent for procstat

To use the procstat plugin, add a `procstat` section in the `metrics_collected` section of the CloudWatch agent configuration file. There are three ways to specify the processes to monitor. You can use only one of these methods, but you can use that method to specify one or more processes to monitor.

- `pid_file`: Selects processes by the names of the process identification number (PID) files they create.
- `exe`: Selects the processes that have process names that match the string that you specify, using regular expression matching rules. The match is a "contains" match, meaning that if you specify `agent` as the term to match, processes with names like `cloudwatchagent` match the term. For more information, see Syntax.
- `pattern`: Selects processes by the command lines used to start the processes. All processes are selected that have command lines matching the specified string using regular expression matching rules. The entire command line is checked, including parameters and options used with the command.

  The match is a "contains" match, meaning that if you specify `-c` as the term to match, processes with parameters like `-config` match the term.

The CloudWatch agent uses only one of these methods, even if you include more than one of the above sections. If you specify more than one section, the CloudWatch agent uses the `pid_file` section if it is present. If not, it uses the `exe` section.

On Linux servers, the strings that you specify in an `exe` or `pattern` section are evaluated as regular expressions. On servers running Windows Server, these strings are evaluated as WMI queries. An example would be `pattern: "%apache%"`. For more information, see LIKE Operator.

Whichever method you use, you can include an optional `metrics_collection_interval` parameter, which specifies how often in seconds to collect those metrics. If you omit this parameter, the default value of 60 seconds is used.

In the examples in the following sections, the `procstat` section is the only section included in the `metrics_collected` section of the agent configuration file. Actual configuration files can also include other sections in `metrics_collected`. For more information, see Manually create or edit the CloudWatch agent configuration file.

Configure with pid_file

The following example `procstat` section monitors the processes that create the PID files `example1.pid` and `example2.pid`. Different metrics are collected from each process. Metrics collected from the process that creates `example2.pid` are collected every 10 seconds, and the metrics collected from the `example1.pid` process are collected every 60 seconds, the default value.

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "pid_file": "/var/run/example1.pid",
                    "measurement": [
                        "cpu_usage",
                        "memory_rss"
                    ]
                },
                {
                    "pid_file": "/var/run/example2.pid",
                    "measurement": [
                        "read_bytes",
                        "read_count",
                        "write_bytes"
                    ],
                    "metrics_collection_interval": 10
                }
            ]
```

```
        }
    }
}
```

Configuring with exe

The following example `procstat` section monitors all processes with names that match the strings `agent` or `plugin`. The same metrics are collected from each process.

```
{
    "metrics": {
        "metrics_collected": {
            "procstat": [
                {
                    "exe": "agent",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                },
                {
                    "exe": "plugin",
                    "measurement": [
                        "cpu_time",
                        "cpu_time_system",
                        "cpu_time_user"
                    ]
                }
            ]
        }
    }
}
```

Configuring with pattern

The following example `procstat` section monitors all processes with command lines that match the strings `config` or `-c`. The same metrics are collected from each process.

```
{
    "metrics": {
        "metrics_collected": {
```

```
            "procstat": [
                {
                    "pattern": "config",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                },
                {
                    "pattern": "-c",
                    "measurement": [
                        "rlimit_memory_data_hard",
                        "rlimit_memory_data_soft",
                        "rlimit_memory_stack_hard",
                        "rlimit_memory_stack_soft"
                    ]
                }
            ]
        }
    }
}
```

Metrics collected by procstat

The following table lists the metrics that you can collect with the `procstat` plugin.

The CloudWatch agent adds `procstat` to the beginning of the following metric names. There is a different syntax depending on whether it was collected from a Linux server or a server running Windows Server. For example, the `cpu_time` metric appears as `procstat_cpu_time` when collected from Linux and as `procstat cpu_time` when collected from Windows Server.

Metric name	Available on	Description
`cpu_time`	Linux	The amount of time that the process uses the CPU. This metric is measured in

Metric name	Available on	Description
		hundredths of a second. Unit: Count
cpu_time_guest	Linux	The amount of time that the process is in guest mode. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_guest_nice	Linux	The amount of time that the process is running in a nice guest. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
cpu_time_idle	Linux	The amount of time that the process is in idle mode. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_iowait	Linux	The amount of time that the process is waiting for I/O operations to complete. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
cpu_time_irq	Linux	The amount of time that the process is servicing interrupts. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_nice	Linux	The amount of time that the process is in nice mode. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
cpu_time_soft_irq	Linux	The amount of time that the process is servicing software interrupts. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_steal	Linux	The amount of time spent running in other operating systems when running in a virtualized environment. This metric is measured in hundredths of a second. Type: Float Unit: None

Metric name	Available on	Description
cpu_time_stolen	Linux, Windows Server	The amount of time that the process is in *stolen time*, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second. Type: Float Unit: None
cpu_time_system	Linux, Windows Server, macOS	The amount of time that the process is in system mode. This metric is measured in hundredths of a second. Type: Float Unit: Count

Metric name	Available on	Description
cpu_time_user	Linux, Windows Server, macOS	The amount of time that the process is in user mode. This metric is measured in hundredths of a second. Unit: Count
cpu_usage	Linux, Windows Server, macOS	The percentage of time that the process is active in any capacity. Unit: Percent
memory_data	Linux, macOS	The amount of memory that the process uses for data. Unit: Bytes
memory_locked	Linux, macOS	The amount of memory that the process has locked. Unit: Bytes

Metric name	Available on	Description
memory_rss	Linux, Windows Server, macOS	The amount of real memory (resident set) that the process is using. Unit: Bytes
memory_stack	Linux, macOS	The amount of stack memory that the process is using. Unit: Bytes
memory_swap	Linux, macOS	The amount of swap memory that the process is using. Unit: Bytes
memory_vms	Linux, Windows Server, macOS	The amount of virtual memory that the process is using. Unit: Bytes

Metric name	Available on	Description
num_fds	Linux	The number of file descriptors that this process has open. Unit: None
num_threads	Linux, Windows, macOS	The number of threads in this process. Unit: None
pid	Linux, Windows Server, macOS	Process identifier (ID). Unit: None

Metric name	Available on	Description
`pid_count`	Linux, Windows Server. macOS	The number of process IDs associated with the process. On Linux servers and macOS computers the full name of this metric is `procstat_lookup_pid_count` and on Windows Server it is `procstat_lookup pid_count` . Unit: None
`read_bytes`	Linux, Windows Server	The number of bytes that the process has read from disks. Unit: Bytes

Metric name	Available on	Description
write_bytes	Linux, Windows Server	The number of bytes that the process has written to disks. Unit: Bytes
read_count	Linux, Windows Server	The number of disk read operations that the process has executed. Unit: None
rlimit_realtime_priority_hard	Linux	The hard limit on the real-time priority that can be set for this process. Unit: None
rlimit_realtime_priority_soft	Linux	The soft limit on the real-time priority that can be set for this process. Unit: None

Metric name	Available on	Description
rlimit_signals_pending_hard	Linux	The hard limit on maximum number of signals that can be queued by this process. Unit: None
rlimit_signals_pending_soft	Linux	The soft limit on maximum number of signals that can be queued by this process. Unit: None
rlimit_nice_priority_hard	Linux	The hard limit on the maximum nice priority that can be set by this process. Unit: None

Metric name	Available on	Description
rlimit_nice_priority_soft	Linux	The soft limit on the maximum nice priority that can be set by this process. Unit: None
rlimit_num_fds_hard	Linux	The hard limit on the maximum number of file descriptors that this process can have open. Unit: None
rlimit_num_fds_soft	Linux	The soft limit on the maximum number of file descriptors that this process can have open. Unit: None

Metric name	Available on	Description
write_count	Linux, Windows Server	The number of disk write operations that the process has executed. Unit: None
involuntary_context_switches	Linux	The number of times that the process was involuntarily context-switched. Unit: None
voluntary_context_switches	Linux	The number of times that the process was context-switched voluntarily. Unit: None
realtime_priority	Linux	The current usage of real-time priority for the process. Unit: None

Metric name	Available on	Description
nice_priority	Linux	The current usage of nice priority for the process. Unit: None
signals_pending	Linux	The number of signals pending to be handled by the process. Unit: None
rlimit_cpu_time_hard	Linux	The hard CPU time resource limit for the process. Unit: None
rlimit_cpu_time_soft	Linux	The soft CPU time resource limit for the process. Unit: None
rlimit_file_locks_hard	Linux	The hard file locks resource limit for the process. Unit: None

Metric name	Available on	Description
rlimit_file_locks_soft	Linux	The soft file locks resource limit for the process. Unit: None
rlimit_memory_data_hard	Linux	The hard resource limit on the process for memory used for data. Unit: Bytes
rlimit_memory_data_soft	Linux	The soft resource limit on the process for memory used for data. Unit: Bytes
rlimit_memory_locked_hard	Linux	The hard resource limit on the process for locked memory. Unit: Bytes

Metric name	Available on	Description
rlimit_memory_locked_soft	Linux	The soft resource limit on the process for locked memory. Unit: Bytes
rlimit_memory_rss_hard	Linux	The hard resource limit on the process for physical memory. Unit: Bytes
rlimit_memory_rss_soft	Linux	The soft resource limit on the process for physical memory. Unit: Bytes
rlimit_memory_stack_hard	Linux	The hard resource limit on the process stack. Unit: Bytes
rlimit_memory_stack_soft	Linux	The soft resource limit on the process stack. Unit: Bytes

Metric name	Available on	Description
`rlimit_memory_vms_hard`	Linux	The hard resource limit on the process for virtual memory. Unit: Bytes
`rlimit_memory_vms_soft`	Linux	The soft resource limit on the process for virtual memory. Unit: Bytes

Viewing process metrics imported by the CloudWatch agent

After importing process metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view process metrics as a time series graph. For more information about setting alarms, see Using Amazon CloudWatch alarms.

To view process metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

b. To sort the table, use the column heading.

c. To filter by resource, choose the resource ID and then choose **Add to search**.

d. To filter by metric, choose the metric name and then choose **Add to search**.

6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Retrieve custom metrics with StatsD

You can retrieve additional custom metrics from your applications or services using the CloudWatch agent with the `StatsD` protocol. StatsD is a popular open-source solution that can gather metrics from a wide variety of applications. StatsD is especially useful for instrumenting your own metrics. For an example of using the CloudWatch agent and StatsD together, see How to better monitor your custom application metrics using Amazon CloudWatch Agent.

`StatsD` is supported on both Linux servers and servers running Windows Server. CloudWatch supports the following `StatsD` format:

```
MetricName:value|type|@sample_rate|#tag1:
  value,tag1...
```

- `MetricName` – A string with no colons, bars, # characters, or @ characters.
- `value` – This can be either integer or float.
- `type` – Specify `c` for counter, `g` for gauge, `ms` for timer, `h` for histogram, or `s` for set.
- `sample_rate` – (Optional) A float between 0 and 1, inclusive. Use only for counter, histogram, and timer metrics. The default value is 1 (sampling 100% of the time).
- `tags` – (Optional) A comma-separated list of tags. `StatsD` tags are similar to dimensions in CloudWatch. Use colons for key/value tags, such as `env:prod`.

You can use any `StatsD` client that follows this format to send the metrics to the CloudWatch agent. For more information about some of the available `StatsD` clients, see the StatsD client page on GitHub.

To collect these custom metrics, add a `"statsd": {}` line to the `metrics_collected` section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it's done for you. For more information, see Create the CloudWatch agent configuration file.

The `StatsD` default configuration works for most users. There are optional fields that you can add to the **statsd** section of the agent configuration file as needed:

- `service_address` – The service address to which the CloudWatch agent should listen. The format is *`ip`*`:`*`port`*. If you omit the IP address, the agent listens on all available interfaces. Only the UDP format is supported, so you don't need to specify a UDP prefix.

  The default value is `:8125`.
- `metrics_collection_interval` – How often in seconds that the `StatsD` plugin runs and collects metrics. The default value is 10 seconds. The range is 1–172,000.
- `metrics_aggregation_interval` – How often in seconds CloudWatch aggregates metrics into single data points. The default value is 60 seconds.

  For example, if `metrics_collection_interval` is 10 and `metrics_aggregation_interval` is 60, CloudWatch collects data every 10 seconds. After each minute, the six data readings from that minute are aggregated into a single data point, which is sent to CloudWatch.

  The range is 0–172,000. Setting `metrics_aggregation_interval` to 0 disables the aggregation of `StatsD` metrics.
- `allowed_pending_messages` – The number of UDP messages that are allowed to queue up. When the queue is full, the StatsD server starts dropping packets. The default value is 10000.
- `drop_original_metrics` – Optional. If you are using the `aggregation_dimensions` field in the `metrics` section to roll up metrics into aggregated results, then by default the agent sends both the aggregated metrics and the original metrics that are separated for each value of the dimension. If you don't want the original metrics to be sent to CloudWatch, you can specify this parameter with a list of metrics. The metrics specified along with this parameter don't have their metrics by dimension reported to CloudWatch. Instead, only the aggregated metrics are reported. This reduces the number of metrics that the agent collects, reducing your costs.

The following is an example of the **statsd** section of the agent configuration file, using the default port and custom collection and aggregation intervals.

```
{
   "metrics":{
      "metrics_collected":{
         "statsd":{
```

```
            "service_address":":8125",
            "metrics_collection_interval":60,
            "metrics_aggregation_interval":300
         }
      }
   }
}
```

Viewing StatsD metrics imported by the CloudWatch agent

After importing StatsD metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view StatsD metrics as a time series graph. For more information about setting alarms, see Using Amazon CloudWatch alarms.

To view StatsD metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

    b. To sort the table, use the column heading.

    c. To filter by resource, choose the resource ID and then choose **Add to search**.

    d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Retrieve custom metrics with collectd

You can retrieve additional metrics from your applications or services using the CloudWatch agent with the collectd protocol, which is supported only on Linux servers. collectd is a popular open-source solution with plugins that can gather system statistics for a wide variety of applications. By combining the system metrics that the CloudWatch agent can already collect with the additional

metrics from collectd, you can better monitor, analyze, and troubleshoot your systems and applications. For more information about collectd, see collectd - The system statistics collection daemon.

You use the collectd software to send the metrics to the CloudWatch agent. For the collectd metrics, the CloudWatch agent acts as the server while the collectd plugin acts as the client.

The collectd software is not installed automatically on every server. On a server running Amazon Linux 2, follow these steps to install collectd

```
sudo amazon-linux-extras install collectd
```

For information about installing collectd on other systems, see the Download page for collectd.

To collect these custom metrics, add a **"collectd": {}** line to the **metrics_collected** section of the agent configuration file. You can add this line manually. If you use the wizard to create the configuration file, it is done for you. For more information, see Create the CloudWatch agent configuration file.

Optional parameters are also available. If you are using collectd and you do not use `/etc/collectd/auth_file` as your **collectd_auth_file**, you must set some of these options.

- **service_address:** The service address to which the CloudWatch agent should listen. The format is `"udp://`*`ip`*`:`*`port`*. The default is `udp://127.0.0.1:25826`.
- **name_prefix:** A prefix to attach to the beginning of the name of each collectd metric. The default is `collectd_`. The maximum length is 255 characters.
- **collectd_security_level:** Sets the security level for network communication. The default is **encrypt**.

  encrypt specifies that only encrypted data is accepted. **sign** specifies that only signed and encrypted data is accepted. **none** specifies that all data is accepted. If you specify a value for **collectd_auth_file**, encrypted data is decrypted if possible.

  For more information, see Client setup and Possible interactions in the collectd Wiki.
- **collectd_auth_file** Sets a file in which user names are mapped to passwords. These passwords are used to verify signatures and to decrypt encrypted network packets. If given, signed data is verified and encrypted packets are decrypted. Otherwise, signed data is accepted without checking the signature and encrypted data cannot be decrypted.

The default is `/etc/collectd/auth_file`.

If **collectd_security_level** is set to **none**, this is optional. If you set **collectd_security_level** to `encrypt` or **sign**, you must specify **collectd_auth_file**.

For the format of the auth file, each line is a user name followed by a colon and any number of spaces followed by the password. For example:

```
user1: user1_password

user2: user2_password
```

- **collectd_typesdb:** A list of one or more files that contain the dataset descriptions. The list must be surrounded by brackets, even if there is just one entry in the list. Each entry in the list must be surrounded by double quotes. If there are multiple entries, separate them with commas. The default on Linux servers is `["/usr/share/collectd/types.db"]`. The default on macOs computers depends on the version of collectd. For example, `["/usr/local/Cellar/collectd/5.12.0/share/collectd/types.db"]`.

  For more information, see https://www.collectd.org/documentation/manpages/types.db.html.
- **metrics_aggregation_interval:** How often in seconds CloudWatch aggregates metrics into single data points. The default is 60 seconds. The range is 0 to 172,000. Setting it to 0 disables the aggregation of collectd metrics.

The following is an example of the collectd section of the agent configuration file.

```
{
   "metrics":{
      "metrics_collected":{
         "collectd":{
            "name_prefix":"My_collectd_metrics_",
            "metrics_aggregation_interval":120
         }
      }
   }
}
```

Viewing collected metrics imported by the CloudWatch agent

After importing collectd metrics into CloudWatch, you can view these metrics as time series graphs, and create alarms that can watch these metrics and notify you if they breach a threshold that you specify. The following procedure shows how to view collectd metrics as a time series graph. For more information about setting alarms, see Using Amazon CloudWatch alarms.

To view collectd metrics in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose the namespace for the metrics collected by the agent. By default, this is **CWAgent**, but you may have specified a different namespace in the CloudWatch agent configuration file.
4. Choose a metric dimension (for example, **Per-Instance Metrics**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

    b. To sort the table, use the column heading.

    c. To filter by resource, choose the resource ID and then choose **Add to search**.

    d. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

Set up and configure Prometheus metrics collection on Amazon EC2 instances

The following sections explain how to install the CloudWatch agent with Prometheus monitoring on EC2 instances, and how to configure the agent to scrape additional targets. It also provides tutorials for setting up sample workloads to use testing with Prometheus monitoring.

Both Linux and Windows instances are supported.

For information about the operating systems supported by the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent

VPC security group requirements

If you are using a VPC, the following requirements apply.

- The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.
- The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- Step 1: Install the CloudWatch agent
- Step 2: Scrape Prometheus sources and import metrics
- Example: Set up Java/JMX sample workloads for Prometheus metric testing

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see Installing the CloudWatch agent.

Step 2: Scrape Prometheus sources and import metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config>in the Prometheus documentation. The other is for the CloudWatch agent configuration.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. A sample configuration file contains the following global configuration lines:

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
- job_name: MY_JOB
  sample_limit: 10000
  file_sd_configs:
```

```
    - files: ["C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_1.yaml",
"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus_sd_2.yaml"]
```

The `global` section specifies parameters that are valid in all configuration contexts. They also serve as defaults for other configuration sections. It contains the following parameters:

- `scrape_interval`— Defines how frequently to scrape targets.
- `scrape_timeout`— Defines how long to wait before a scrape request times out.

The `scrape_configs` section specifies a set of targets and parameters that define how to scrape them. It contains the following parameters:

- `job_name`— The job name assigned to scraped metrics by default.
- `sample_limit`— Per-scrape limit on the number of scraped samples that will be accepted.
- `file_sd_configs`— List of file service discovery configurations. It reads a set of files containing a list of zero or more static configs. The `file_sd_configs` section contains a `files` parameter which defines patterns for files from which target groups are extracted.

The CloudWatch agent supports the following service discovery configuration types.

`static_config` Allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

The following is a sample static config to scrape Prometheus metrics from a local host. Metrics can also be scraped from other servers if the Prometheus port is open to the server where the agent runs.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_sd_1.yaml
- targets:
    - 127.0.0.1:9404
  labels:
    key1: value1
    key2: value2
```

This example contains the following parameters:

- `targets`— The targets scraped by the static config.
- `labels`— Labels assigned to all metrics that are scraped from the targets.

`ec2_sd_config` Allows retrieving scrape targets from Amazon EC2 instances. The following is a sample `ec2_sd_config` to scrape Prometheus metrics from a list of EC2 instances. The Prometheus ports of these instances have to open to the server where the CloudWatch agent runs. The IAM role for the EC2 instance where the CloudWatch agent runs must include the `ec2:DescribeInstance` permission. For example, you could attach the managed policy **AmazonEC2ReadOnlyAccess** to the instance running the CloudWatch agent.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: MY_JOB
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - i-98765432109876543
              - i-12345678901234567
```

This example contains the following parameters:

- `region`— The AWS Region where the target EC2 instance is. If you leave this blank, the Region from the instance metadata is used.
- `port`— The port to scrape metrics from.
- `filters`— Optional filters to use to filter the instance list. This example filters based on EC2 instance IDs. For more criteria that you can filter on, see DescribeInstances.

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file includes `prometheus` sections under both `logs` and `metrics_collected`. It includes the following parameters.

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics.

- **prometheus_config_path**— specifies the Prometheus scrape configuration file path.
- **emf_processor**— specifies the embedded metric format processor configuration. For more information about embedded metric format, see Embedding metrics within logs.

  The `emf_processor` section can contain the following parameters:

  - **metric_declaration_dedup**— It set to true, the de-duplication function for the embedded metric format metrics is enabled.
  - **metric_namespace**— Specifies the metric namespace for the emitted CloudWatch metrics.
  - **metric_unit**— Specifies the metric name:metric unit map. For information about supported metric units, see MetricDatum.
  - **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
    - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
    - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.
    - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
    - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

The following is an example CloudWatch agent configuration for Prometheus.

```
{
   "logs":{
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-cluster",
            "log_group_name":"Prometheus",
            "prometheus_config_path":"C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
\prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent-Prometheus",
```

```
                "metric_unit":{
                   "jvm_threads_current": "Count",
                   "jvm_gc_collection_seconds_sum": "Milliseconds"
                },
                "metric_declaration":[
                   {
                      "source_labels":[
                         "job", "key2"
                      ],
                      "label_matcher":"MY_JOB;^value2",
                      "dimensions":[
                         [
                            "key1", "key2"
                         ],
                         [
                            "key2"
                         ]
                      ],
                      "metric_selectors":[
                         "^jvm_threads_current$",
                         "^jvm_gc_collection_seconds_sum$"
                      ]
                   }
                ]
             }
          }
       }
    }
}
```

The previous example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of the label `job` is `MY_JOB`
- The value of the label `key2` is `value2`
- The Prometheus metrics `jvm_threads_current` and `jvm_gc_collection_seconds_sum` contains both `job` and `key2` labels.

The log event that is sent includes the following highlighted section.

```
{
```

```
    "CloudWatchMetrics": [
        {
            "Metrics": [
                {
                    "Unit": "Count",
                    "Name": "jvm_threads_current"
                },
                {
                    "Unit": "Milliseconds",
                    "Name": "jvm_gc_collection_seconds_sum"
                }
            ],
            "Dimensions": [
                [
                    "key1",
                    "key2"
                ],
                [
                    "key2"
                ]
            ],
            "Namespace": "CWAgent-Prometheus"
        }
    ],
    "ClusterName": "prometheus-cluster",
    "InstanceId": "i-0e45bd06f196096c8",
    "Timestamp": "1607966368109",
    "Version": "0",
    "host": "EC2AMAZ-PDDOIUM",
    "instance": "127.0.0.1:9404",
    "jvm_threads_current": 2,
    "jvm_gc_collection_seconds_sum": 0.006000000000000002,
    "prom_metric_type": "gauge",
    ...
}
```

Example: Set up Java/JMX sample workloads for Prometheus metric testing

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

The CloudWatch agent can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Hjava, and Tomcat (Catalina), from a JMX exporter on EC2 instances.

Step 1: Install the CloudWatch agent

The first step is to install the CloudWatch agent on the EC2 instance. For instructions, see Installing the CloudWatch agent.

Step 2: Start the Java/JMX workload

The next step is to start the Java/JMX workload.

First, download the latest JMX exporter jar file from the following location: prometheus/jmx_exporter.

Use the jar for your sample application

The example commands in the following sections use `SampleJavaApplication-1.0-SNAPSHOT.jar` as the jar file. Replace these parts of the commands with the jar for your application.

Prepare the JMX exporter configuration

The `config.yaml` file is the JMX exporter configuration file. For more information, see Configuration in the JMX exporter documentation.

Here is a sample configuration for Java and Tomcat.

```
---
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
```

```
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

Start the Java application with the Prometheus exporter

Start the sample application. This will emit Prometheus metrics to port 9404. Be sure to replace the entry point `com.gubupt.sample.app.App` with the correct information for your sample java application.

On Linux, enter the following command.

```
$ nohup java -javaagent:./jmx_prometheus_javaagent-0.14.0.jar=9404:./config.yaml -cp
  ./SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App &
```

On Windows, enter the following command.

```
PS C:\> java -javaagent:.\jmx_prometheus_javaagent-0.14.0.jar=9404:.\config.yaml -cp  .
\SampleJavaApplication-1.0-SNAPSHOT.jar com.gubupt.sample.app.App
```

Verify the Prometheus metrics emission

Verify that Prometheus metrics are being emitted.

On Linux, enter the following command.

```
$ curl localhost:9404
```

On Windows, enter the following command.

```
PS C:\> curl  http://localhost:9404
```

Sample output on Linux:

```
StatusCode        : 200
StatusDescription : OK
Content           : # HELP jvm_classes_loaded The number of classes that are currently
 loaded in the JVM
                    # TYPE jvm_classes_loaded gauge
                    jvm_classes_loaded 2526.0
                    # HELP jvm_classes_loaded_total The total number of class...
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 71908
                    Content-Type: text/plain; version=0.0.4; charset=utf-8
                    Date: Fri, 18 Dec 2020 16:38:10 GMT

                    # HELP jvm_classes_loaded The number of classes that are
 currentl...
Forms             : {}
Headers           : {[Content-Length, 71908], [Content-Type, text/plain; version=0.0.4;
 charset=utf-8], [Date, Fri, 18
                    Dec 2020 16:38:10 GMT]}
Images            : {}
InputFields       : {}
```

```
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 71908
```

Step 3: Configure the CloudWatch agent to scrape Prometheus metrics

Next, set up the Prometheus scrape configuration in the CloudWatch agent configuration file.

To set up the Prometheus scrape configuration for the Java/JMX example

1. Set up the configuration for `file_sd_config` and `static_config`.

   On Linux, enter the following command.

   ```
   $ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus.yaml
   global:
     scrape_interval: 1m
     scrape_timeout: 10s
   scrape_configs:
     - job_name: jmx
       sample_limit: 10000
       file_sd_configs:
         - files: [ "/opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml" ]
   ```

   On Windows, enter the following command.

   ```
   PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus.yaml
   global:
     scrape_interval: 1m
     scrape_timeout: 10s
   scrape_configs:
     - job_name: jmx
       sample_limit: 10000
       file_sd_configs:
         - files: [ "C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\
   \prometheus_file_sd.yaml" ]
   ```

2. Set up the scrape targets configuration.

   On Linux, enter the following command.

   ```
   $ cat /opt/aws/amazon-cloudwatch-agent/var/prometheus_file_sd.yaml
   ```

```
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: linux
```

On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
    os: windows
```

3. Set up the Prometheus scrape configuration by `ec2_sc_config`. Replace *your-ec2-instance-id* with the correct EC2 instance ID.

   On Linux, enter the following command.

```
$ cat .\prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    ec2_sd_configs:
      - region: us-east-1
        port: 9404
        filters:
          - name: instance-id
            values:
              - your-ec2-instance-id
```

   On Windows, enter the following command.

```
PS C:\ProgramData\Amazon\AmazonCloudWatchAgent> cat prometheus_file_sd.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    application: sample_java_app
```

```
    os: windows
```

4. Set up the CloudWatch agent configuration. First, navigate to the correct directory. On Linux, it is `/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json`. On Windows, it is `C:\ProgramData\Amazon\AmazonCloudWatchAgent\cwagent-config.json`.

   The following is a sample configuration with Java/JHX Prometheus metrics defined. Be sure to replace *`path-to-Prometheus-Scrape-Configuration-file`* with the correct path.

```
{
  "agent": {
    "region": "us-east-1"
  },
  "logs": {
    "metrics_collected": {
      "prometheus": {
        "cluster_name": "my-cluster",
        "log_group_name": "prometheus-test",
        "prometheus_config_path": "path-to-Prometheus-Scrape-Configuration-file",
        "emf_processor": {
          "metric_declaration_dedup": true,
          "metric_namespace": "PrometheusTest",
          "metric_unit":{
            "jvm_threads_current": "Count",
            "jvm_classes_loaded": "Count",
            "java_lang_operatingsystem_freephysicalmemorysize": "Bytes",
            "catalina_manager_activesessions": "Count",
            "jvm_gc_collection_seconds_sum": "Seconds",
            "catalina_globalrequestprocessor_bytesreceived": "Bytes",
            "jvm_memory_bytes_used": "Bytes",
            "jvm_memory_pool_bytes_used": "Bytes"
          },
          "metric_declaration": [
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["instance"]],
              "metric_selectors": [
                "^jvm_threads_current$",
                "^jvm_classes_loaded$",
                "^java_lang_operatingsystem_freephysicalmemorysize$",
                "^catalina_manager_activesessions$",
                "^jvm_gc_collection_seconds_sum$",
```

```
                "^catalina_globalrequestprocessor_bytesreceived$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["area"]],
              "metric_selectors": [
                "^jvm_memory_bytes_used$"
              ]
            },
            {
              "source_labels": ["job"],
              "label_matcher": "^jmx$",
              "dimensions": [["pool"]],
              "metric_selectors": [
                "^jvm_memory_pool_bytes_used$"
              ]
            }
          ]
        }
      }
    },
    "force_flush_interval": 5
  }
}
```

5. Restart the CloudWatch agent by entering one of the following commands.

   On Linux, enter the following command.

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
   config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/var/cwagent-config.json
   ```

   On Windows, enter the following command.

   ```
   & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1"
    -a fetch-config -m ec2 -s -c file:C:\ProgramData\Amazon\AmazonCloudWatchAgent
   \cwagent-config.json
   ```

Viewing the Prometheus metrics and logs

You can now view the Java/JMX metrics being collected.

To view the metrics for your sample Java/JMX workload

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **PrometheusTest** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **prometheus-test**.

Configure CloudWatch agent service and environment names for related entities

The CloudWatch agent can send metrics and logs with entity data to support the Explore related pane in the CloudWatch console. The service name or environment name can be configured by the CloudWatch Agent JSON configuration.

> **Note**
>
> The agent configuration may be overridden. For details about how the agent decides what data to send for related entities, see Using the CloudWatch agent with related telemetry.

For metrics, it can be configured at the agent, metrics, or plugin level. For logs it can be configured at the agent, logs, or file level. The most specific configuration is always used. For example if the configuration exists at the agent level and metrics level, then metrics will use the metric configuration, and anything else (logs) will use the agent configuration. The following example shows different ways to configure the service name and environment name.

```
{
  "agent": {
    "service.name": "agent-level-service",
    "deployment.environment": "agent-level-environment"
  },

  "metrics": {
    "service.name": "metric-level-service",
     "deployment.environment": "metric-level-environment",
```

```
    "metrics_collected": {
      "statsd": {
        "service.name": "statsd-level-service",
        "deployment.environment": "statsd-level-environment",
      },
      "collectd": {
        "service.name": "collectdd-level-service",
        "deployment.environment": "collectd-level-environment",
      }
    }

  },

  "logs": {
    "service.name": "log-level-service",
    "deployment.environment": "log-level-environment",

    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-
agent.log",
            "log_group_name": "amazon-cloudwatch-agent.log",
            "log_stream_name": "amazon-cloudwatch-agent.log",

            "service.name": "file-level-service",
            "deployment.environment": "file-level-environment"
          }
        ]
      }
    }

  }
}
```

Starting the CloudWatch agent

You can start the CloudWatch agent using either Systems Manager Run Command or the command line.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

Start the CloudWatch agent using the command line on Amazon EC2

Follow these steps to use the command line to start the CloudWatch agent on on Amazon EC2.

For information about setting up the agent on a system that has security-enhanced Linux (SELinux) enabled, see Set up the CloudWatch agent with security-enhanced Linux (SELinux).

To use the command line to start the CloudWatch agent on on Amazon EC2

1. Copy the agent configuration file that you want to use to the server where you're going to run the agent. Note the pathname where you copy it to.
2. In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

   Enter one of the following commands. Replace *`configuration-file-path`* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

   On an EC2 instance running Linux, enter the following command.

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
   config -m ec2 -s -c file:configuration-file-path
   ```

   On an on-premises server running Linux, enter the following:

   ```
   sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
   config -m onPremise -s -c file:configuration-file-path
   ```

   On an EC2 instance running Windows Server, enter the following from the PowerShell console:

   ```
   & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
   a fetch-config -m ec2 -s -c file:configuration-file-path
   ```

   On an on-premises server running Windows Server, enter the following from the PowerShell console:

   ```
   & "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
   a fetch-config -m onPremise -s -c file:configuration-file-path
   ```

Start the CloudWatch agent on an on-premises server

Follow these steps to start the CloudWatch agent on an on-premises server.

To use SSM Agent to start the CloudWatch agent on an on-premises server

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, select the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, select the instance where you installed the agent.
6. In the **Action** list, choose **configure**.
7. In the **Mode** list, choose **onPremise**.
8. In the **Optional Configuration Location** box, enter the name of the agent configuration file that you created with the wizard and stored in the Parameter Store.
9. Choose **Run**.

   The agent starts with the configuration you specified in the configuration file.

To use the command line to start the CloudWatch agent on an on-premises server

- In this command, `-a fetch-config` causes the agent to load the latest version of the CloudWatch agent configuration file, and `-s` starts the agent.

  Linux: If you saved the configuration file in the Systems Manager Parameter Store, enter the following:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Linux: If you saved the configuration file on the local computer, enter the following command. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m onPremise -s -c file:configuration-file-path
```

Windows Server: If you saved the agent configuration file in Systems Manager Parameter Store, enter the following from the PowerShell console:

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c ssm:configuration-parameter-store-name
```

Windows Server: If you saved the agent configuration file on the local computer, enter the following from the PowerShell console. Replace *configuration-file-path* with the path to the agent configuration file. This file is called `config.json` if you created it with the wizard, and might be called `amazon-cloudwatch-agent.json` if you created it manually.

```
& "C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1" -
a fetch-config -m onPremise -s -c file:configuration-file-path
```

Metrics collected by the CloudWatch agent

You can collect metrics from servers by installing the CloudWatch agent on the server. You can install the agent on both Amazon EC2 instances and on-premises servers. You can also install the agent on computers running Linux, Windows Server, or macOS. If you install the agent on an Amazon EC2 instance, the metrics the agent collects are in addition to the metrics enabled by default on Amazon EC2 instances. For information about installing the CloudWatch agent on an instance, see Collect metrics, logs, and traces using the CloudWatch agent. You can use this section to learn about metrics the CloudWatch agent collects.

Metrics collected by the CloudWatch agent on Windows Server instances

On a server running Windows Server, installing the CloudWatch agent enables you to collect the metrics associated with the counters in Windows Performance Monitor. The CloudWatch metric names for these counters are created by putting a space between the object name and the counter

name. For example, the `% Interrupt Time` counter of the `Processor` object is given the metric name `Processor % Interrupt Time` in CloudWatch. For more information about Windows Performance Monitor counters, see the Microsoft Windows Server documentation.

The default namespace for metrics collected by the CloudWatch agent is `CWAgent`, although you can specify a different namespace when you configure the agent.

Metrics collected by the CloudWatch agent on Linux and macOS instances

The following table lists the metrics that you can collect with the CloudWatch agent on Linux servers and macOS computers.

Metric	Description
`cpu_time_active`	The amount of time that the CPU is active in any capacity. This metric is measured in hundredths of a second. Unit: None
`cpu_time_guest`	The amount of time that the CPU is running a virtual CPU for a guest operating system. This metric is measured in hundredths of a second. Unit: None
`cpu_time_guest_nice`	The amount of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. This metric is measured in hundredths of a second. Unit: None
`cpu_time_idle`	The amount of time that the CPU is idle. This metric is measured in hundredths of a second. Unit: None

Metric	Description
cpu_time_iowait	The amount of time that the CPU is waiting for I/O operations to complete. This metric is measured in hundredths of a second. Unit: None
cpu_time_irq	The amount of time that the CPU is servicing interrupts. This metric is measured in hundredths of a second. Unit: None
cpu_time_nice	The amount of time that the CPU is in user mode with low-priority processes, which can easily be interrupted by higher-priority processes. This metric is measured in hundredths of a second. Unit: None
cpu_time_softirq	The amount of time that the CPU is servicing software interrupts. This metric is measured in hundredths of a second. Unit: None
cpu_time_steal	The amount of time that the CPU is in *stolen time*, which is time spent in other operating systems in a virtualized environment. This metric is measured in hundredths of a second. Unit: None
cpu_time_system	The amount of time that the CPU is in system mode. This metric is measured in hundredths of a second. Unit: None

Metric	Description
cpu_time_user	The amount of time that the CPU is in user mode. This metric is measured in hundredths of a second. Unit: None
cpu_usage_active	The percentage of time that the CPU is active in any capacity. Unit: Percent
cpu_usage_guest	The percentage of time that the CPU is running a virtual CPU for a guest operating system. Unit: Percent
cpu_usage_guest_nice	The percentage of time that the CPU is running a virtual CPU for a guest operating system, which is low-priority and can be interrupted by other processes. Unit: Percent
cpu_usage_idle	The percentage of time that the CPU is idle. Unit: Percent
cpu_usage_iowait	The percentage of time that the CPU is waiting for I/O operations to complete. Unit: Percent
cpu_usage_irq	The percentage of time that the CPU is servicing interrupts. Unit: Percent

Metric	Description
cpu_usage_nice	The percentage of time that the CPU is in user mode with low-priority processes, which higher-priority processes can easily interrupt. Unit: Percent
cpu_usage_softirq	The percentage of time that the CPU is servicing software interrupts. Unit: Percent
cpu_usage_steal	The percentage of time that the CPU is in *stolen time*, or time spent in other operating systems in a virtualized environment. Unit: Percent
cpu_usage_system	The percentage of time that the CPU is in system mode. Unit: Percent
cpu_usage_user	The percentage of time that the CPU is in user mode. Unit: Percent
disk_free	Free space on the disks. Unit: Bytes
disk_inodes_free	The number of available index nodes on the disk. Unit: Count
disk_inodes_total	The total number of index nodes reserved on the disk. Unit: Count

Metric	Description
`disk_inodes_used`	The number of used index nodes on the disk. Unit: Count
`disk_total`	Total space on the disks, including used and free. Unit: Bytes
`disk_used`	Used space on the disks. Unit: Bytes
`disk_used_percent`	The percentage of total disk space that is used. Unit: Percent
`diskio_iops_in_progress`	The number of I/O requests that have been issued to the device driver but have not yet completed. Unit: Count
`diskio_io_time`	The amount of time that the disk has had I/O requests queued. Unit: Milliseconds The only statistic that should be used for this metric is `Sum`. Do not use `Average`.
`diskio_reads`	The number of disk read operations. Unit: Count The only statistic that should be used for this metric is `Sum`. Do not use `Average`.

Metric	Description
diskio_read_bytes	The number of bytes read from the disks. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_read_time	The amount of time that read requests have waited on the disks. Multiple read requests waiting at the same time increase the number. For example, if 5 requests all wait for an average of 100 milliseconds, 500 is reported. Unit: Milliseconds The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_writes	The number disk write operations. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
diskio_write_bytes	The number of bytes written to the disks. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
diskio_write_time	The amount of time that write requests have waited on the disks. Multiple write requests waiting at the same time increase the number. For example, if 8 requests all wait for an average of 1000 milliseconds, 8000 is reported. Unit: Milliseconds The only statistic that should be used for this metric is Sum. Do not use Average.
ethtool_bw_in_allowance_exceeded	The number of packets queued and/or dropped because the inbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
ethtool_bw_out_allowance_exceeded	The number of packets queued and/or dropped because the outbound aggregate bandwidth exceeded the maximum for the instance. This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
ethtool_conntrack_allowance_exceeded	The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance. This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None
ethtool_linklocal_allowance_exceeded	The number of packets dropped because the PPS of the traffic to local proxy services exceeded the maximum for the network interface. This impacts traffic to the DNS service, the Instance Metadata Service, and the Amazon Time Sync Service. This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics Unit: None

Metric	Description
ethtool_pps_allowance_exceeded	The number of packets queued and/or dropped because the bidirectional PPS exceeded the maximum for the instance. This metric is collected only if you have listed it in the ethtool subsection of the metrics_collected section of the CloudWatch agent configuration file. For more information, see Collect network performance metrics. Unit: None
mem_active	The amount of memory that has been used in some way during the last sample period. Unit: Bytes
mem_available	The amount of memory that is available and can be given instantly to processes. Unit: Bytes
mem_available_percent	The percentage of memory that is available and can be given instantly to processes. Unit: Percent
mem_buffered	The amount of memory that is being used for buffers. Unit: Bytes
mem_cached	The amount of memory that is being used for file caches. Unit: Bytes

Metric	Description
mem_free	The amount of memory that isn't being used. Unit: Bytes
mem_inactive	The amount of memory that hasn't been used in some way during the last sample period Unit: Bytes
mem_shared	The amount of memory that is being shared between processes. Unit: Bytes
mem_total	The total amount of memory. Unit: Bytes
mem_used	The amount of memory currently in use. Unit: Bytes
mem_used_percent	The percentage of memory currently in use. Unit: Percent
net_bytes_recv	The number of bytes received by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.
net_bytes_sent	The number of bytes sent by the network interface. Unit: Bytes The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
net_drop_in	The number of packets received by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_drop_out	The number of packets transmitted by this network interface that were dropped. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_err_in	The number of receive errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_err_out	The number of transmit errors detected by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
net_packets_sent	The number of packets sent by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.

Metric	Description
net_packets_recv	The number of packets received by this network interface. Unit: Count The only statistic that should be used for this metric is Sum. Do not use Average.
netstat_tcp_close	The number of TCP connections with no state. Unit: Count
netstat_tcp_close_wait	The number of TCP connections waiting for a termination request from the client. Unit: Count
netstat_tcp_closing	The number of TCP connections that are waiting for a termination request with acknowledgment from the client. Unit: Count
netstat_tcp_established	The number of TCP connections established. Unit: Count
netstat_tcp_fin_wait1	The number of TCP connections in the FIN_WAIT1 state during the process of closing a connection. Unit: Count
netstat_tcp_fin_wait2	The number of TCP connections in the FIN_WAIT2 state during the process of closing a connection. Unit: Count

Metric	Description
netstat_tcp_last_ack	The number of TCP connections waiting for the client to send acknowledgment of the connectio n termination message. This is the last state right before the connection is closed down. Unit: Count
netstat_tcp_listen	The number of TCP ports currently listening for a connection request. Unit: Count
netstat_tcp_none	The number of TCP connections with inactive clients. Unit: Count
netstat_tcp_syn_sent	The number of TCP connections waiting for a matching connection request after having sent a connection request. Unit: Count
netstat_tcp_syn_recv	The number of TCP connections waiting for connection request acknowledgment after having sent and received a connection request. Unit: Count
netstat_tcp_time_wait	The number of TCP connections currently waiting to make sure that the client received the acknowled gment of its connection termination request. Unit: Count
netstat_udp_socket	The number of current UDP connections. Unit: Count

Metric	Description
processes_blocked	The number of processes that are blocked. Unit: Count
processes_dead	The number of processes that are dead, indicated by the X state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_idle	The number of processes that are idle (sleeping for more than 20 seconds). Available only on FreeBSD instances. Unit: Count
processes_paging	The number of processes that are paging, indicated by the W state code on Linux. This metric is not collected on macOS computers. Unit: Count
processes_running	The number of processes that are running, indicated by the R state code. Unit: Count
processes_sleeping	The number of processes that are sleeping, indicated by the S state code. Unit: Count
processes_stopped	The number of processes that are stopped, indicated by the T state code. Unit: Count

Metric	Description
processes_total	The total number of processes on the instance. Unit: Count
processes_total_threads	The total number of threads making up the processes. This metric is available only on Linux instances. This metric is not collected on macOS computers. Unit: Count
processes_wait	The number of processes that are paging, indicated by the W state code on FreeBSD instances. This metric is available only on FreeBSD instances, and is not available on Linux, Windows Server, or macOS instances. Unit: Count
processes_zombies	The number of zombie processes, indicated by the Z state code. Unit: Count
swap_free	The amount of swap space that isn't being used. Unit: Bytes
swap_used	The amount of swap space currently in use. Unit: Bytes
swap_used_percent	The percentage of swap space currently in use. Unit: Percent

Definitions of memory metrics collected by the CloudWatch agent

When the CloudWatch agent collects memory metrics, the source is the host's memory management subsystem. For example, the Linux kernel exposes OS-maintained data in `/proc`. For memory, the data is in `/proc/meminfo`.

Each different operating system and architecture has different calculations of the resources that are used by processes. For more information, see the following sections.

During each collection interval, the CloudWatch agent on each instance collects the instance resources and calculates the resources being used by all processes which are running in that instance. This information is reported back to CloudWatch metrics. You can configure the length of the collection interval in the CloudWatch agent configuration file. For more information, see CloudWatch agent configuration file: Agent section.

The following list explains how the memory metrics that the CloudWatch agent collects are defined.

- **Active Memory**– Memory that is being used by a process. In other words, the memory used by current running apps.
- **Available Memory**– The memory that can be instantly given to the processes without the system going into swap (also known as virtual memory).
- **Buffer Memory**– The data area shared by hardware devices or program processes that operate at different speeds and priorities.
- **Cached Memory**– Stores program instructions and data that are used repeatedly in the operation of programs that the CPU is likely to need next.
- **Free Memory**– Memory that is not being used at all and is readily available. It is completely free for the system to be used when needed.
- **Inactive Memory**– Pages that have not been accessed "recently".
- **Total Memory**– The size of the actual physical memory RAM.
- **Used Memory**– Memory that is currently in use by programs and processes.

Topics

- Linux: Metrics collected and calculations used
- macOS: Metrics collected and calculations used
- Windows: Metrics collected

- Example: Calculating memory metrics on Linux

Linux: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Buffered (Bytes)
- Cached (Bytes)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Used memory = Total Memory - Free Memory - Cached memory - Buffer memory

Total memory = Used Memory + Free Memory + Cached memory + Buffer memory

macOS: Metrics collected and calculations used

Metrics collected and units:

- Active (Bytes)
- Available (Bytes)
- Available Percent (Percent)
- Free (Bytes)
- Inactive (Bytes)
- Total (Bytes)
- Used (Bytes)
- Used Percent (Percent)

Available memory = Free Memory + Inactive memory

Used memory = Total Memory - Available memory

Total memory = Available Memory + Used Memory

Windows: Metrics collected

The metrics collected on Windows hosts are listed below. All of these metrics have `None` for `Unit`.

- Available bytes
- Cache Faults/sec
- Page Faults/sec
- Pages/sec

There are no calculations used for Windows metrics because the CloudWatch agent parses events from performance counters.

Example: Calculating memory metrics on Linux

As an example, suppose that entering the **`cat /proc/meminfo`** command on a Linux host shows the following results:

```
MemTotal:       3824388 kB
MemFree:         462704 kB
MemAvailable:   2157328 kB
Buffers:         126268 kB
Cached:         1560520 kB
SReclaimable:    289080 kB>
```

In this example, the CloudWatch agent will collect the following values. All the values that the CloudWatch agent collects and reports are in bytes.

- `mem_total`: 3916173312 bytes
- `mem_available`: 2209103872 bytes (MemFree + Cached)
- `mem_free`: 473808896 bytes
- `mem_cached`: 1893990400 bytes (`cached` + `SReclaimable`)
- `mem_used`: 1419075584 bytes (`MemTotal` – (`MemFree` + `Buffers` + (`Cached` + `SReclaimable`)))
- `mem_buffered`: 129667072 bytes

- `mem_available_percent`: 56.41%
- `mem_used_percent`: 36.24% (`mem_used` / `mem_total`) * 100

Using the CloudWatch agent with related telemetry

Metrics and logs that are sent to CloudWatch can include an optional entity to correlate telemetry. Entities are used in the Explore related pane. The CloudWatch agent sends entities with a service name and environment name included.

The agent chooses the service name and environment name from the following data.

Service name

The agent chooses the service name from the following options, in priority order:

- **Application Signals instrumentation** – The agent sends the service name used by Application Signals. This can be overwritten by changing the `OTEL_SERVICE_NAME` environment variable used by supported OpenTelemetry instrumentation libraries.
- **CloudWatch agent configuration** – You can configure the agent to use a specific service name. This can be configured at the agent, plugin, metrics, logs, or log file level.
- **Kubernetes workload name** – For Kubernetes workloads, the agent sends the name of the workload for the corresponding pod, in the following priority order.
  - Deployment name
  - ReplicaSet name
  - StatefulSet name
  - DaemonSet name
  - CronJob name
  - Job name
  - Pod name
  - Container name
- **Resource tags from instance metadata** – For Amazon EC2 workloads, the agent sends the a name from tags, in the following order.
  - service
  - application
  - app

You must setup instance metadata for the agent to be able to access tags.

- **Default** – If no other service name is found, the agent will send the name `Unknown`.

Environment name

The agent chooses the environment name from the following options, in priority order:

- **Application Signals instrumentation** – The agent sends the environment name used by Application Signals. This can be overwritten by setting a `deployment.environment` environment variable used by supported OpenTelemetry instrumentation libraries. For example, applications may set the environment variable `OTEL_RESOURCE_ATTRIBUTES=deployment.environment=MyEnvironment`.
- **CloudWatch agent configuration** – You can configure the agent to use a specific environment name. This can be configured at the agent, plugin, metrics, logs, or log file level.
- **Cluster name and workspace** – For Amazon EKS, `eks:`*`cluster-name`*`/`*`Namespace`*. For native Kubernetes running on Amazon EC2, `k8s:`*`cluster-name`*`/`*`Namespace`*.
- **Resource tags from instance metadata** – For Amazon EC2 workloads, the agent can will use the `AutoScalingGroup` tag.

  You must setup instance metadata for the agent to be able to access tags.
- By default, Amazon EC2 instances that aren't running Kubernetes will get the environment name `ec2:default`.

Common scenarios with the CloudWatch agent

This section provides you with different scenarios that outline how to complete common configuration and customization tasks for the CloudWatch agent.

Topics

- Running the CloudWatch agent as a different user
- How the CloudWatch agent handles sparse log files
- Adding custom dimensions to metrics collected by the CloudWatch agent
- Aggregating or rolling up metrics collected by the CloudWatch agent
- Collecting high-resolution metrics with the CloudWatch agent

- Sending metrics, logs, and traces to a different account
- Timestamp differences between the CloudWatch agent and the earlier CloudWatch Logs agent
- Appending OpenTelemetry collector configuration files

Running the CloudWatch agent as a different user

On Linux servers, the CloudWatch runs as the root user by default. To have the agent run as a different user, use the `run_as_user` parameter in the `agent` section in the CloudWatch agent configuration file. This option is available only on Linux servers.

If you're already running the agent with the root user and want to change to using a different user, use one of the following procedures.

To run the CloudWatch agent as a different user on an EC2 instance running Linux

1. Download and install a new CloudWatch agent package.
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Provide credentials for this user in one of these ways:
   - If the file `.aws/credentials` exists in the home directory of the root user, you must create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be /home/*username*/`.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see Install the CloudWatch agent using AWS Systems Manager.
   - If the file `.aws/credentials` does not exist in the home directory of the root user, you can do one of the following:
     - Create a credentials file for the user you are going to use to run the CloudWatch agent. This credentials file will be /home/*username*/`.aws/credentials`. Then set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see Install the CloudWatch agent using AWS Systems Manager.
     - Instead of creating a credentials file, attach an IAM role to the instance. The agent uses this role as the credential provider.
4. In the CloudWatch agent configuration file, add the following line in the `agent` section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see Create the CloudWatch agent configuration file

5. Give the user the required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
6. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m ec2 -s -c file:configuration-file-path
```

To run the CloudWatch agent as a different user on an on-premises server running Linux

1. Download and install a new CloudWatch agent package.
2. Create a new Linux user or use the default user named `cwagent` that the RPM or DEB file created.
3. Store the credentials of this user to a path that the user can access, such as `/home/username/.aws/credentials`.
4. Set the value of the `shared_credential_file` parameter in `common-config.toml` to the pathname of the credential file. For more information, see Install the CloudWatch agent using AWS Systems Manager.
5. In the CloudWatch agent configuration file, add the following line in the `agent` section:

```
"run_as_user": "username"
```

Make other modifications to the configuration file as needed. For more information, see Create the CloudWatch agent configuration file

6. Give the user required permissions. The user must have Read (r) permissions for the log files to be collected, and must have Execute (x) permission on every directory in the log files' path.
7. Start the agent with the configuration file that you just modified.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-
config -m ec2 -s -c file:configuration-file-path
```

How the CloudWatch agent handles sparse log files

Sparse files are files with both empty blocks and real contents. A sparse file uses disk space more efficiently by writing brief information representing the empty blocks to disk instead of the actual null bytes which makes up the block. This makes the actual size of a sparse file usually much smaller than its apparent size.

However, the CloudWatch agent doesn't treat sparse files differently than it treats normal files. When the agent reads a sparse file, the empty blocks are treated as "real" blocks filled with null bytes. Because of this, the CloudWatch agent publishes as many bytes as the apparent size of a sparse file to CloudWatch.

Configuring the CloudWatch agent to publish a sparse file can cause higher than expected CloudWatch costs, so we recommend not to do so. For example, `/var/logs/lastlog` in Linux is usually a very sparse file, and we recommend that you don't publish it to CloudWatch.

Adding custom dimensions to metrics collected by the CloudWatch agent

To add custom dimensions such as tags to metrics collected by the agent, add the `append_dimensions` field to the section of the agent configuration file that lists those metrics.

For example, the following example section of the configuration file adds a custom dimension named `stackName` with a value of `Prod` to the `cpu` and `disk` metrics collected by the agent.

```
"cpu":{
  "resources":[
    "*"
  ],
  "measurement":[
    "cpu_usage_guest",
    "cpu_usage_nice",
    "cpu_usage_idle"
  ],
  "totalcpu":false,
  "append_dimensions":{
    "stackName":"Prod"
  }
},
"disk":{
  "resources":[
    "/",
    "/tmp"
```

```
  ],
  "measurement":[
    "total",
    "used"
  ],
  "append_dimensions":{
    "stackName":"Prod"
  }
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Aggregating or rolling up metrics collected by the CloudWatch agent

To aggregate or roll up metrics collected by the agent, add an `aggregation_dimensions` field to the section for that metric in the agent configuration file.

For example, the following configuration file snippet rolls up metrics on the `AutoScalingGroupName` dimension. The metrics from all instances in each Auto Scaling group are aggregated and can be viewed as a whole.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"]]
}
```

To roll up along the combination of each `InstanceId` and `InstanceType` dimensions in addition to rolling up on the Auto Scaling group name, add the following.

```
"metrics": {
  "cpu":{...}
  "disk":{...}
  "aggregation_dimensions" : [["AutoScalingGroupName"], ["InstanceId", "InstanceType"]]
}
```

To roll up metrics into one collection instead, use `[]`.

```
"metrics": {
  "cpu":{...}
```

```
  "disk":{...}
  "aggregation_dimensions" : [[]]
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Collecting high-resolution metrics with the CloudWatch agent

The `metrics_collection_interval` field specifies the time interval for the metrics collected, in seconds. By specifying a value of less than 60 for this field, the metrics are collected as high-resolution metrics.

For example, if your metrics should all be high-resolution and collected every 10 seconds, specify 10 as the value for `metrics_collection_interval` under the `agent` section as a global metrics collection interval.

```
"agent": {
  "metrics_collection_interval": 10
}
```

Alternatively, the following example sets the `cpu` metrics to be collected every second, and all other metrics are collected every minute.

```
"agent":{
  "metrics_collection_interval": 60
},
"metrics":{
  "metrics_collected":{
    "cpu":{
      "resources":[
        "*"
      ],
      "measurement":[
        "cpu_usage_guest"
      ],
      "totalcpu":false,
      "metrics_collection_interval": 1
    },
    "disk":{
      "resources":[
        "/",
```

```
        "/tmp"
      ],
      "measurement":[
        "total",
        "used"
      ]
    }
  }
}
```

Remember that any time you change the agent configuration file, you must restart the agent to have the changes take effect.

Sending metrics, logs, and traces to a different account

To have the CloudWatch agent send the metrics, logs, or traces to a different account, specify a `role_arn` parameter in the agent configuration file on the sending server. The `role_arn` value specifies an IAM role in the target account that the agent uses when sending data to the target account. This role enables the sending account to assume a corresponding role in the target account when delivering the metrics or logs to the target account.

You can also specify separate `role_arn` strings in the agent configuration file: one to use when sending metrics, another for sending logs, and another for sending traces.

The following example of part of the `agent` section of the configuration file sets the agent to use `CrossAccountAgentRole` when sending data to a different account.

```
{
  "agent": {
    "credentials": {
      "role_arn": "arn:aws:iam::123456789012:role/CrossAccountAgentRole"
    }
  },
  .....
}
```

Alternatively, the following example sets different roles for the sending account to use for sending metrics, logs, and traces:

```
"metrics": {
    "credentials": {
```

```
     "role_arn": "RoleToSendMetrics"
    },
    "metrics_collected": {....
```

```
"logs": {
    "credentials": {
    "role_arn": "RoleToSendLogs"
    },
    ....
```

Policies needed

When you specify a `role_arn` in the agent configuration file, you must also make sure the IAM roles of the sending and target accounts have certain policies. The roles in both the sending and target accounts should have `CloudWatchAgentServerPolicy`. For more information about assigning this policy to a role, see Prerequisites.

The role in the sending account also must include the following policy. You add this policy on the **Permissions** tab in the IAM console when you edit the role.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": [
                "arn:aws:iam::111122223333:role/agent-role-in-target-account"
            ]
        }
    ]
}
```

The role in the target account must include the following policy so that it recognizes the IAM role used by the sending account. You add this policy on the **Trust relationships** tab in the IAM console

when you edit the role. This role is the role specified in *agent-role-in-target-account* in the policy used by the sending account.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:role/role-in-sender-account"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Timestamp differences between the CloudWatch agent and the earlier CloudWatch Logs agent

The CloudWatch agent supports a different set of symbols for timestamp formats, compared to the earlier CloudWatch Logs agent. These differences are shown in the following table.

Symbols supported by both agents	Symbols supported only by CloudWatch agent	Symbols supported only by earlier CloudWatch Logs agent
%A, %a, %b, %B, %d, %f, %H, %l, %m, %M, %p, %S, %y, %Y, %Z, %z	%-d, %-l, %-m, %-M, %-S	%c,%j, %U, %W, %w

For more information about the meanings of the symbols supported by the new CloudWatch agent, see CloudWatch Agent Configuration File: Logs Section in the *Amazon CloudWatch User*

Guide. For information about symbols supported by the CloudWatch Logs agent, see Agent Configuration File in the *Amazon CloudWatch Logs User Guide*.

Appending OpenTelemetry collector configuration files

The CloudWatch agent supports supplemental OpenTelemetry collector configuration files alongside its own configuration files. This feature allows you to use CloudWatch agent features such as CloudWatch Application Signals or Container Insights through the CloudWatch agent configuration and bring in your existing OpenTelemetry collector configuration with a single agent.

To prevent merge conflicts with pipelines automatically created by CloudWatch agent, add a custom suffix to each of the components and pipelines in your OpenTelemetry collector configuration.

```
receivers:
  otlp/custom-suffix:
    protocols:
      http:

exporters:
  awscloudwatchlogs/custom-suffix:
    log_group_name: "test-group"
    log_stream_name: "test-stream"

service:
  pipelines:
    logs/custom-suffix:
      receivers: [otlp/custom-suffix]
      exporters: [awscloudwatchlogs/custom-suffix]
```

To configure the CloudWatch agent, start the CloudWatch agent using the `fetch-config` option and specify the CloudWatch agent's configuration file. CloudWatch agent requires at least one CloudWatch agent configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -c
 file:/tmp/agent.json -s
```

Next, use the `append-config` option while specifying the OpenTelemetry collector configuration file.

```
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a append-config -c
 file:/tmp/otel.yaml -s
```

The agent merges the two configuration files on start up and logs the resolved configuration.

The following OpenTelemetry components are available for you to configure in your appended YAML configuration. Use the component type name shown here as the key in your YAML.

Component type	Available components
Receivers	`otlp`, `prometheus` , `statsd`, `collectd`, `jmx`, `hostmetrics` , `filelog`, `tcplog`, `udplog`, `jaeger`, `zipkin`, `kafka`, `kubeletstats`
Processors	`batch`, `memory_limiter` , `filter`, `attributes` , `resource`, `resourcedetection` , `metricstransform` , `transform` , `cumulativetodelta` , `deltatocumulative` , `deltatorate` , `groupbyattrs` , `groupbytrace` , `k8sattributes` , `metricsgeneration` , `metricstarttime` , `probabilistic_sampler` , `span`, `tail_sampling`
Exporters	`otlphttp`, `awsemf`, `awscloudwatchlogs` , `awsxray`, `prometheusremotewrite` , `debug`
Extensions	`sigv4auth` , `headers_setter` , `file_storage` , `health_check` , `pprof`, `zpages`

The CloudWatch agent only supports writing telemetry to AWS destinations.

For YAML examples that send each signal to the CloudWatch OpenTelemetry Protocol (OTLP) endpoints, see Configuration examples.

CloudWatch agent credentials preference

This section outlines the credentials provider chain the CloudWatch agent uses to obtain credentials when communicating with other AWS services and APIs. The ordering is as follows:

> **Note**
>
> The preferences listed in numbers two through five are the same preference order as defined in the AWS SDK. For more information, see Specifying Credentials in the SDK documentation.

1. Shared config and credentials files as defined in the CloudWatch agent's `common-config.toml` file. For more information, see Install the CloudWatch agent using AWS Systems Manager.
2. AWS SDK environment variables

> **Important**
>
> On Linux, if you run the CloudWatch agent using the `amazon-cloudwatch-agent-ctl` script, the script starts the agent as a `systemd` service. In this case, environment variables such as `HOME`, `AWS_ACCESS_KEY_ID`, and `AWS_SECRET_ACCESS_KEY` are not accessible by the agent.

3. Shared configuration and credentials files found in `$HOME/%USERPROFILE%`

> **Note**
>
> The CloudWatch agent looks for `.aws/credentials` in `$HOME` on Linux and macOS and in `%USERPROFILE%` on Windows. These environment variables resolve to the home directory of the user the agent runs as: `root` on Linux and macOS (or the `run_as_user` if configured), and `LocalSystem` on Windows.
> Unlike the AWS SDK, the CloudWatch agent does not have fallback methods to determine the home directory if the environment variables are inaccessible. This difference in behavior is to maintain backwards compatibility with earlier implementations of the AWS SDK.
> Furthermore, unlike with the shared credentials found in `common-config.toml`, if the AWS SDK-derived shared credentials expire and are rotated, the renewed credentials are not automatically picked up by the CloudWatch agent and require a restart of the agent to do so.

4. An AWS Identity and Access Management role for tasks if an application is present that uses an Amazon Elastic Container Service task definition or a RunTask API operation.

5. An instance profile attached to an Amazon EC2 instance.

As a best practice, we recommend that you specify credentials in the following order when you use the CloudWatch agent.

1. Use IAM roles for tasks if your application uses an Amazon Elastic Container Service task definition or a RunTask API operation.
2. Use IAM roles if your application runs on an Amazon EC2 instance.
3. Use the CloudWatch agent `common-config.toml` file to specify the credentials file. This credentials file is the same one used by other AWS SDKs and the AWS CLI. If you're already using a shared credentials file, you can also use it for this purpose. If you provide it by using the CloudWatch agent's `common-config.toml` file, you ensure that the agent will consume rotated credentials when they expire and get replaced without requiring you to restart the agent.
4. Use environment variables. Setting environment variables is useful if you're doing development work on a computer other than an Amazon EC2 instance.

> **Note**
>
> If you send telemetry to a different account as explained in Sending metrics, logs, and traces to a different account, the CloudWatch agent uses the credentials provider chain described in this section to obtain the initial set of credentials. It then uses those credentials when assuming the IAM role specified by `role_arn` in the CloudWatch agent configuration file.

Troubleshooting the CloudWatch agent

You can use the information in this section to troubleshoot issues you might encounter with the CloudWatch agent.

When encountering issues with the CloudWatch agent, you can use the `AWSSupport-TroubleshootCloudWatchAgent` automation runbook. The AWS troubleshooting tool can:

- Verify IAM permissions and instance profiles
- Check agent status and analyze logs
- Test endpoint connectivity

- Automatically collect and upload relevant logs to Amazon S3

For detailed information on the AWS troubleshooting tool, see Support Automation Workflow (SAW) Runbook - Troubleshoot CloudWatch agent.

Topics

- CloudWatch agent command line parameters
- Install the CloudWatch agent using Run Command fails
- The CloudWatch agent won't start
- Verify that the CloudWatch agent is running
- The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region
- The CloudWatch agent won't start on Windows Server
- Where are the metrics?
- The CloudWatch agent takes a long time to run in a container or logs a hop limit error
- I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console
- The CloudWatch agent is using the wrong credentials
- CloudWatch agent files and locations
- Finding information about CloudWatch agent versions
- Logs generated by the CloudWatch agent
- Stopping and restarting the CloudWatch agent

CloudWatch agent command line parameters

To see the full list of parameters supported by the CloudWatch agent, enter the following at the command line at a computer where you have it installed:

```
amazon-cloudwatch-agent-ctl -help
```

Install the CloudWatch agent using Run Command fails

To install the CloudWatch agent using Systems Manager Run Command, the SSM Agent on the target server must be version 2.2.93.0 or later of the SSM Agent agent. If your SSM Agent isn't the correct version, you might see errors that include the following messages:

```
no latest version found for package AmazonCloudWatchAgent on platform linux
```

```
failed to download installation package reliably
```

For information about updating your SSM Agent version, see Installing and Configuring SSM Agent in the *AWS Systems Manager User Guide*.

The CloudWatch agent won't start

If the CloudWatch agent fails to start, there might be an issue in your configuration. Configuration information is logged in the `configuration-validation.log` file. This file is located in `/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` on Linux servers and in `$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log` on servers running Windows Server.

Verify that the CloudWatch agent is running

You can query the CloudWatch agent to find whether it's running or stopped. You can use AWS Systems Manager to do this remotely. You can also use the command line, but only to check the local server.

To query the status of the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose the button next to **AmazonCloudWatch-ManageAgent**.
5. In the **Action** list, choose **status**.
6. For **Optional Configuration Source** choose **default** and keep **Optional Configuration Location** blank.
7. In the **Target** area, choose the instance to check.

8. Choose **Run**.

If the agent is running, the output resembles the following.

```
{
       "status": "running",
       "starttime": "2017-12-12T18:41:18",
       "version": "1.73.4"
}
```

If the agent is stopped, the `"status"` field displays `"stopped"`.

To query the status of the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
   status
  ```

  On a server running Windows Server, enter the following in PowerShell as an administrator:

  ```
  & $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
   ec2 -a status
  ```

The CloudWatch agent won't start, and the error mentions an Amazon EC2 Region

If the agent doesn't start and the error message mentions an Amazon EC2 Region endpoint, you might have configured the agent to need access to the Amazon EC2 endpoint without granting that access.

For example, if you specify a value for the `append_dimensions` parameter in the agent configuration file that depends on Amazon EC2 metadata and you use proxies, you must make sure that the server can access the endpoint for Amazon EC2. For more information about these endpoints, see Amazon Elastic Compute Cloud (Amazon EC2) in the *Amazon Web Services General Reference*.

The CloudWatch agent won't start on Windows Server

On Windows Server, you might see the following error:

```
Start-Service : Service 'Amazon CloudWatch Agent (AmazonCloudWatchAgent)' cannot be
 started due to the following
error: Cannot start service AmazonCloudWatchAgent on computer '.'.
At C:\Program Files\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1:113
 char:12
+     $svc | Start-Service
+            ~~~~~~~~~~~~~
    + CategoryInfo          : OpenError:
 (System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
   ServiceCommandException
    + FullyQualifiedErrorId :
 CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

To fix this, first make sure that the server service is running. This error can be seen if the agent tries to start when the server service isn't running.

If the server service is already running, the following may be the issue. On some Windows Server installations, the CloudWatch agent takes more than 30 seconds to start. Because Windows Server, by default, allows only 30 seconds for services to start, this causes the agent to fail with an error similar to the following:

To fix this issue, increase the service timeout value. For more information, see A service does not start, and events 7000 and 7011 are logged in the Windows event log.

Where are the metrics?

If the CloudWatch agent has been running but you can't find metrics collected by it in the AWS Management Console or the AWS CLI, confirm that you're using the correct namespace. By default, the namespace for metrics collected by the agent is `CWAgent`. You can customize this namespace using the `namespace` field in the `metrics` section of the agent configuration file. If you don't see the metrics that you expect, check the configuration file to confirm the namespace being used.

When you first download the CloudWatch agent package, the agent configuration file is `amazon-cloudwatch-agent.json`. This file is in the directory where you ran the configuration wizard, or you might have moved it to a different directory. If you use the configuration wizard, the agent configuration file output from the wizard is named `config.json`. For more information about the configuration file, including the `namespace` field, see CloudWatch agent configuration file: Metrics section.

The CloudWatch agent takes a long time to run in a container or logs a hop limit error

When you run the CloudWatch agent as a container service and want to add Amazon EC2 metric dimensions to all metrics collected by the agent, you might see the following errors in version v1.247354.0 of the agent:

```
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Unable to retrieve Instance
 Metadata Tags. This plugin must only be used on an EC2 instance.
2022-06-07T03:36:11Z E! [processors.ec2tagger] ec2tagger: Please increase hop limit
 to 2 by following this document https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
configuring-instance-metadata-options.html#configuring-IMDS-existing-instances.
2022-06-07T03:36:11Z E! [telegraf] Error running agent: could not initialize processor
 ec2tagger: EC2MetadataRequestError: failed to get EC2 instance identity document
caused by: EC2MetadataError: failed to make EC2Metadata request
        status code: 401, request id:
caused by:
```

You might see this error if the agent tries to get metadata from IMDSv2 inside a container without an appropriate hop limit. In versions of the agent earlier than v1.247354.0, you can experience this issue without seeing the log message.

To solve this, increase the hop limit to 2 by following the instructions in Configure the instance metadata options.

I updated my agent configuration but don't see the new metrics or logs in the CloudWatch console

If you update your CloudWatch agent configuration file, the next time that you start the agent, you need to use the **`fetch-config`** option. For example, if you stored the updated file on the local computer, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -
s -m ec2 -c file:configuration-file-path
```

The CloudWatch agent is using the wrong credentials

If the CloudWatch agent is using unexpected credentials, the agent may be resolving credentials from a different source than you expect. The agent follows a credentials provider chain that

differs from the AWS SDK in some cases. For more information, see CloudWatch agent credentials preference.

CloudWatch agent files and locations

The following table lists the files installed by and used with the CloudWatch agent, along with their locations on servers running Linux or Windows Server.

File	Linux location	Windows Server location
The control script that controls starting, stopping, and restarting the agent.	`/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl` or `/usr/bin/amazon-cloudwatch-agent-ctl`	`$Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1`
The log file the agent writes to. You might need to attach this when contacting AWS Support.	`/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` or `/var/log/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.log`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log`
Agent configuration validation file.	`/opt/aws/amazon-cloudwatch-agent/logs/configuration-validation.log` or `/var/log/amazon/amazon-cloudwatch-agent/configuration-validation.log`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\configuration-validation.log`
The JSON file used to configure the agent immediately after the wizard	`/opt/aws/amazon-cloudwatch-agent/bin/config.json`	`$Env:ProgramFiles\Amazon\AmazonCloud`

File	Linux location	Windows Server location
creates it. For more information, see Create the CloudWatch agent configuration file.		WatchAgent\config.json
The JSON file used to configure the agent if this configuration file has been downloaded from Parameter Store.	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.json	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.json
The TOML file used to specify Region and credential information to be used by the agent, overriding system defaults.	/opt/aws/amazon-cloudwatch-agent/etc/common-config.toml or /etc/amazon/amazon-cloudwatch-agent/common-config.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\common-config.toml
The TOML file that contains the converted contents of the JSON configuration file. The amazon-cloudwatch-agent-ctl script generates this file. Users should not directly modify this file. It can be useful for verifying that JSON to TOML translation was successful.	/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.toml	$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.toml

File	Linux location	Windows Server location
The YAML file that contains the converted contents of the JSON configuration file. The `amazon-cloudwatch-agent-ctl` script generates this file. You should not directly modify this file. This file can be useful for verifying that the JSON to YAML translation was successful.	`/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.yaml or /etc/amazon/amazon-cloudwatch-agent/amazon-cloudwatch-agent.yaml`	`$Env:ProgramData\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent.yaml`

Finding information about CloudWatch agent versions

To find the version number of the CloudWatch agent on a Linux server, enter the following command:

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

To find the version number of the CloudWatch agent on Windows Server, enter the following command:

```
& $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m ec2
 -a status
```

> **Note**
>
> Using this command is the correct way to find the version of the CloudWatch agent. If you use **Programs and Features** in the Control Panel, you will see an incorrect version number.

You can also download a README file about the latest changes to the agent, and a file that indicates the version number that is currently available for download. These files are in the following locations:

- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/RELEASE_NOTES or https://amazoncloudwatch-agent-*us-east-1*.s3.*us-east-1*.amazonaws.com/info/latest/RELEASE_NOTES
- https://amazoncloudwatch-agent.s3.amazonaws.com/info/latest/CWAGENT_VERSION or https://amazoncloudwatch-agent-*us-east-1*.s3.*us-east-1*.amazonaws.com/info/latest/CWAGENT_VERSION

Logs generated by the CloudWatch agent

The agent generates a log while it runs. This log includes troubleshooting information. This log is the amazon-cloudwatch-agent.log file. This file is located in /opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log on Linux servers and in $Env:ProgramData\Amazon\AmazonCloudWatchAgent\Logs\amazon-cloudwatch-agent.log on servers running Windows Server.

You can configure the agent to log additional details in the amazon-cloudwatch-agent.log file. In the agent configuration file, in the agent section, set the debug field to true, then reconfigure and restart the CloudWatch agent. To disable the logging of this extra information, set the debug field to false. Then, reconfigure and restart the agent. For more information, see Manually create or edit the CloudWatch agent configuration file.

In versions 1.247350.0 and later of the CloudWatch agent, you can optionally set the aws_sdk_log_level field in the agent section of the agent configuration file to one or more of the following options. Separate multiple options with the | character.

- LogDebug
- LogDebugWithSigning
- LogDebugWithHTTPBody
- LogDebugRequestRetries
- LogDebugWithEventStreamBody

For more information about these options, see LogLevelType.

Stopping and restarting the CloudWatch agent

You can manually stop the CloudWatch agent using either AWS Systems Manager or the command line.

To stop the CloudWatch agent using Run Command

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Run Command**.

   -or-

   If the AWS Systems Manager home page opens, scroll down and choose **Explore Run Command**.
3. Choose **Run command**.
4. In the **Command document** list, choose **AmazonCloudWatch-ManageAgent**.
5. In the **Targets** area, choose the instance where you installed the CloudWatch agent.
6. In the **Action** list, choose **stop**.
7. Keep **Optional Configuration Source** and **Optional Configuration Location** blank.
8. Choose **Run**.

To stop the CloudWatch agent locally using the command line

- On a Linux server, enter the following:

  ```
  sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -m ec2 -a
   stop
  ```

  On a server running Windows Server, enter the following in PowerShell as an administrator:

  ```
  & $Env:ProgramFiles\Amazon\AmazonCloudWatchAgent\amazon-cloudwatch-agent-ctl.ps1 -m
   ec2 -a stop
  ```

To restart the agent, follow the instructions in (Optional) Modify the common configuration and named profile for CloudWatch agent.

Embedding metrics within logs

The CloudWatch embedded metric format allows you to generate custom metrics asynchronously in the form of logs written to CloudWatch Logs. You can embed custom metrics alongside detailed log event data, and CloudWatch automatically extracts the custom metrics so that you can

visualize and alarm on them, for real-time incident detection. Additionally, the detailed log events associated with the extracted metrics can be queried using CloudWatch Logs Insights to provide deep insights into the root causes of operational events.

Embedded metric format helps you generate actionable custom metrics from ephemeral resources such as Lambda functions and containers. By using the embedded metric format to send logs from these ephemeral resources, you can now easily create custom metrics without having to instrument or maintain separate code, while gaining powerful analytical capabilities on your log data.

No setup is required to use the embedded metric format. Either structure your logs by following the Embedded metric format specification, or generate them using our client libraries and send them to CloudWatch Logs using the PutLogEvents API or the CloudWatch agent.

To generate metrics from logs with embedded metric format, you need the `logs:PutLogEvents` permission but you don't need to also have the `cloudwatch:PutMetricData` permission.

Charges are incurred for logs ingestion and archival, and custom metrics that are generated. For more information, see Amazon CloudWatch Pricing.

> **Note**
>
> Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see Dimensions.

> **Note**
>
> Embedded metric format ensures at least one time delivery of metrics extracted from log events, while duplicate metric values may occasionally occur.

The following topics describe how to publish logs using the embedded metric format, view your metrics and logs in the console, and set alarms on metrics created with the embedded metric format.

Topics

- Publishing logs with the embedded metric format
- Viewing your metrics and logs in the console
- Setting alarms on metrics created with the embedded metric format

Publishing logs with the embedded metric format

You can generate embedded metric format logs using the following methods:

- Generate and send the logs by using the open-sourced client libraries.
- Manually generate the logs using the embedded metric format specification, and then use the CloudWatch agent or the PutLogEvents API to send the logs.

The following topics provide more information about embedded metrics.

Topics

- Creating logs in embedded metric format using the client libraries
- Specification: Embedded metric format
- Using the PutLogEvents API to send manually-created embedded metric format logs
- Using the CloudWatch agent to send embedded metric format logs
- Using the embedded metric format with AWS Distro for OpenTelemetry

Creating logs in embedded metric format using the client libraries

You can use the open-sourced client libraries that Amazon provides to create embedded metric format logs. Full examples for different setups can be found in our client libraries under **/ examples**. The libraries and the instructions for how to use them are located on Github.

- Node.Js

> **Note**
>
> For Node.js, versions 4.1.1+, 3.0.2+, 2.0.7+ are required for use with the Lambda JSON log format. Using previous versions in such Lambda environments will lead to metric loss. For more information, see Accessing Amazon CloudWatch logs for AWS Lambda.

- Python
- Java
- C#

Client libraries are meant to work out of the box with the CloudWatch agent. Generated embedded metric format logs are sent to the CloudWatch agent, which then aggregates and publishes them to CloudWatch Logs for you.

> **Note**
>
> When using Lambda, no agent is required to send the logs to CloudWatch. Anything logged to STDOUT is sent to CloudWatch Logs through the Lambda Logging Agent.

Specification: Embedded metric format

The CloudWatch embedded metric format is a JSON specification used to instruct CloudWatch Logs to automatically extract metric values embedded in structured log events. You can use CloudWatch to graph and create alarms on the extracted metric values. This section describes embedded metric format specification conventions and the embedded metric format document structure.

> **Note**
>
> Embedded metric format ensures at least one time delivery of metrics extracted from log events, while duplicate metric values may occasionally occur.

Embedded metric format specification conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this format specification are to be interpreted as described in Key Words RFC2119.

The terms "JSON", "JSON text", "JSON value", "member", "element", "object", "array", "number", "string", "boolean", "true", "false", and "null" in this format specification are to be interpreted as defined in JavaScript Object Notation RFC8259.

> **Note**
>
> If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

Embedded metric format document structure

This section describes the structure of an embedded metric format document. Embedded metric format documents are defined in JavaScript Object Notation RFC8259.

Unless otherwise noted, objects defined by this specification MUST NOT contain any additional members. Members not recognized by this specification MUST be ignored. Members defined in this specification are case-sensitive.

The embedded metric format is subject to the same limits as standard CloudWatch Logs events and are limited to a maximum size of 1 MB.

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `AWS/Logs` namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Root node

The LogEvent message MUST be a valid JSON object with no additional data at the beginning or end of the LogEvent message string. For more information about the LogEvent structure, see InputLogEvent.

Embedded metric format documents MUST contain the following top-level member on the root node. This is a Metadata object object.

```
{
 "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

The root node MUST contain all Target members members defined by the references in the MetricDirective object.

The root node MAY contain any other members that are not included in the above requirements. The values of these members MUST be valid JSON types.

Metadata object

The `_aws` member can be used to represent metadata about the payload that informs downstream services how they should process the LogEvent. The value MUST be an object and MUST contain the following members:

- **CloudWatchMetrics**— An array of MetricDirective object used to instruct CloudWatch to extract metrics from the root node of the LogEvent.

```
{
  "_aws": {
    "CloudWatchMetrics": [ ... ]
  }
}
```

- **Timestamp**— A number representing the time stamp used for metrics extracted from the event. Values MUST be expressed as the number of milliseconds after Jan 1, 1970 00:00:00 UTC. The time stamp MUST be no more than 14 days in the past and no more than 2 hours in the future. CloudWatch does not publish metrics that have a time stamp outside this range.

```
{
  "_aws": {
    "Timestamp": 1559748430481
  }
}
```

MetricDirective object

The MetricDirective object instructs downstream services that the LogEvent contains metrics that will be extracted and published to CloudWatch. MetricDirectives MUST contain the following members:

- **Namespace**— A string representing the CloudWatch namespace for the metric. The namespace MUST contain 1 to 255 printable ASCII characters (0x20 through 0x7E) and MUST NOT contain only whitespace. If the namespace is not valid, CloudWatch does not publish any metrics in that namespace.
- **Dimensions**— A DimensionSet array.

- **Metrics**— An array of MetricDefinition objects. A log event MUST NOT contain more than 100 MetricDefinition objects in total, counted across all of its MetricDirectives. If a log event exceeds this limit, CloudWatch does not extract any metrics from it.

DimensionSet array

A DimensionSet is an array of strings containing the dimension keys that will be applied to all metrics in the document. The values within this array MUST also be members on the root-node—referred to as the Target members

A DimensionSet MUST NOT contain more than 30 dimension keys. A DimensionSet MAY be empty.

Each dimension key has the following constraints:

- It MUST contain 1 to 250 printable ASCII characters.
- It MUST NOT contain only whitespace.
- It MUST NOT start with a colon (`:`).

The target member MUST have a string value. This value MUST contain 1 to 1024 printable ASCII characters and MUST NOT contain only whitespace. The target member defines a dimension that will be published as part of the metric identity. Every DimensionSet used creates a new metric in CloudWatch. For more information about dimensions, see Dimension and Dimensions.

If a DimensionSet or a dimension it references doesn't meet these requirements, CloudWatch doesn't publish the metrics for that DimensionSet. The other DimensionSets in the MetricDirective are not affected.

```
{
 "_aws": {
   "CloudWatchMetrics": [
     {
       "Dimensions": [ [ "functionVersion" ] ],
       ...
     }
   ]
 },
 "functionVersion": "$LATEST"
}
```

> **Note**
>
> Be careful when configuring your metric extraction as it impacts your custom metric usage and corresponding bill. If you unintentionally create metrics based on high-cardinality dimensions (such as `requestId`), the embedded metric format will by design create a custom metric corresponding to each unique dimension combination. For more information, see Dimensions.

MetricDefinition object

A MetricDefinition is an object that MUST contain the following member:

- **Name**— A string Reference values to a metric Target members. Metric targets MUST be either a numeric value or an array of numeric values. The name MUST contain 1 to 255 printable ASCII characters and MUST NOT contain only whitespace. If the name is not valid, CloudWatch does not publish that metric.

A MetricDefinition object MAY contain the following members:

- **Unit**— An OPTIONAL string value representing the unit of measure for the corresponding metric. Values SHOULD be valid CloudWatch metric units. For information about valid units, see MetricDatum. If a value is not provided, then a default value of NONE is assumed.
- **StorageResolution**— An OPTIONAL integer value representing the storage resolution for the corresponding metric. Setting this to 1 specifies this metric as a high-resolution metric, so that CloudWatch stores the metric with sub-minute resolution down to one second. Setting this to 60 specifies this metric as standard-resolution, which CloudWatch stores at 1-minute resolution. Values SHOULD be valid CloudWatch supported resolutions, 1 or 60. If a value is not provided, then a default value of 60 is assumed. CloudWatch stores the metric at high resolution only when the value is 1. For example, CloudWatch treats `1.0` as 60.

  For more information about high-resolution metrics, see High-resolution metrics.

> **Note**
>
> If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

```
{
  "_aws": {
    "CloudWatchMetrics": [
      {
        "Metrics": [
          {
            "Name": "Time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ],
        ...
      }
    ]
  },
  "Time": 1
}
```

Reference values

Reference values are string values that reference Target members members on the root node. These references should NOT be confused with the JSON Pointers described in RFC6901. Target values cannot be nested.

Target members

Valid targets MUST be members on the root node and cannot be nested objects. For example, a _reference_ value of `"A.a"` MUST match the following member:

```
{ "A.a" }
```

It MUST NOT match the nested member:

```
{ "A": { "a" } }
```

Valid values of target members depend on what is referencing them. A metric target MUST be a numeric value or an array of numeric values. Numeric values MUST be in the range -2^{360} to 2^{360}. If a value is outside this range, CloudWatch does not publish that metric. Numeric array metric targets MUST NOT have more than 100 members. If a numeric array exceeds this limit, CloudWatch does not extract any metrics from the log event. A dimension target MUST have a string value.

Embedded metric format example and JSON schema

The following is a valid example of embedded metric format.

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "lambda-function-metrics",
        "Dimensions": [["functionVersion"]],
        "Metrics": [
          {
            "Name": "time",
            "Unit": "Milliseconds",
            "StorageResolution": 60
          }
        ]
      }
    ]
  },
  "functionVersion": "$LATEST",
  "time": 100,
  "requestId": "989ffbf8-9ace-4817-a57c-e4dd734019ee"
}
```

You can use the following schema to validate embedded metric format documents. The schema checks the MetricDefinition limit for each MetricDirective only. It does not check the total across all MetricDirectives in a log event.

```
{
    "type": "object",
    "title": "Root Node",
    "required": [
        "_aws"
    ],
    "properties": {
        "_aws": {
            "$id": "#/properties/_aws",
            "type": "object",
            "title": "Metadata",
            "required": [
                "Timestamp",
```

```
                "CloudWatchMetrics"
            ],
            "properties": {
                "Timestamp": {
                    "$id": "#/properties/_aws/properties/Timestamp",
                    "type": "integer",
                    "title": "The Timestamp Schema",
                    "examples": [
                        1565375354953
                    ]
                },
                "CloudWatchMetrics": {
                    "$id": "#/properties/_aws/properties/CloudWatchMetrics",
                    "type": "array",
                    "title": "MetricDirectives",
                    "items": {
                        "$id": "#/properties/_aws/properties/CloudWatchMetrics/items",
                        "type": "object",
                        "title": "MetricDirective",
                        "required": [
                            "Namespace",
                            "Dimensions",
                            "Metrics"
                        ],
                        "properties": {
                            "Namespace": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Namespace",
                                "type": "string",
                                "title": "CloudWatch Metrics Namespace",
                                "examples": [
                                    "MyApp"
                                ],
                                "pattern": "^[ -~]*[!-~][ -~]*$",
                                "minLength": 1,
                                "maxLength": 255
                            },
                            "Dimensions": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Dimensions",
                                "type": "array",
                                "title": "The Dimensions Schema",
                                "minItems": 1,
                                "items": {
```

```
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items",
                                    "type": "array",
                                    "title": "DimensionSet",
                                    "minItems": 0,
                                    "maxItems": 30,
                                    "items": {
                                        "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Dimensions/items/items",
                                        "type": "string",
                                        "title": "DimensionReference",
                                        "examples": [
                                            "Operation"
                                        ],
                                        "pattern": "^(?:[!-9;-~]| +[!-~])[ -~]*$",
                                        "minLength": 1,
                                        "maxLength": 250
}
                                }
                            },
                            "Metrics": {
                                "$id": "#/properties/_aws/properties/CloudWatchMetrics/
items/properties/Metrics",
                                "type": "array",
                                "title": "MetricDefinitions",
                                "maxItems": 100,
                                "items": {
                                    "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items",
                                    "type": "object",
                                    "title": "MetricDefinition",
                                    "required": [
                                        "Name"
                                    ],
                                    "properties": {
                                        "Name": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Name",
                                            "type": "string",
                                            "title": "MetricName",
                                            "examples": [
                                                "ProcessingLatency"
                                            ],
                                            "pattern": "^[ -~]*[!-~][ -~]*$",
```

```
                                            "minLength": 1,
                                            "maxLength": 255
                                        },
                                        "Unit": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/Unit",
                                            "type": "string",
                                            "title": "MetricUnit",
                                            "examples": [
                                                "Milliseconds"
                                            ],
                                            "pattern": "^(Seconds|Microseconds|
Milliseconds|Bytes|Kilobytes|Megabytes|Gigabytes|Terabytes|Bits|Kilobits|Megabits|
Gigabits|Terabits|Percent|Count|Bytes\\/Second|Kilobytes\\/Second|Megabytes\\/Second|
Gigabytes\\/Second|Terabytes\\/Second|Bits\\/Second|Kilobits\\/Second|Megabits\\/
Second|Gigabits\\/Second|Terabits\\/Second|Count\\/Second|None)$"
                                        },
                                        "StorageResolution": {
                                            "$id": "#/properties/_aws/properties/
CloudWatchMetrics/items/properties/Metrics/items/properties/StorageResolution",
                                            "type": "integer",
                                            "title": "StorageResolution",
                                            "examples": [
                                                60
                                            ]
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Entity information in EMF format

When you publish logs to Amazon CloudWatch using Embedded Metric Format (EMF), you can include entity information in the log event. This section describes how to specify entity information and how CloudWatch processes this information.

Entity types

When no Entity is specified with the `PutLogEvents` request, CloudWatch will look for entity information in the EMF log content:

- **Service-type entities**

  Required fields: `Service` and `Environment`
- **Resource-type entities**

  Required fields: `ResourceType` and `Identifier`

Platform attributes

CloudWatch automatically determines the platform type based on these attributes:

- **Kubernetes (K8s):**

  Required: `K8s.Cluster`

  Optional: `K8s.Namespace`, `K8s.Workload`, `K8s.Node`, `K8s.Pod`, `EC2.InstanceId`, `EC2.AutoScalingGroup`
- **Amazon EKS**

  Required: `EKS.Cluster`

  Optional: `K8s.Namespace`, `K8s.Workload`, `K8s.Node`, `K8s.Pod`, `EC2.InstanceId`
- **Amazon ECS:**

  Required: `ECS.Cluster`

  Optional: `ECS.Service`, `ECS.Task`
- **Amazon EC2**

  Required: `EC2.InstanceId`

  Optional: `EC2.AutoScalingGroup`
- **Lambda:**

  Required: `Lambda.Function`

- **Generic hosts:**

  Required: `Host`

Example EMF log format

```
{
    "_aws": {
        "CloudWatchMetrics": [
            {
                "Metrics": [
                    {"Name": "RequestLatency", "Unit": "Milliseconds"}
                ],
                "Namespace": "MyApplication"
            }
        ]
    },
    "Service": "PaymentService",
    "Environment": "Production",
    "K8s.Cluster": "main-cluster",
    "K8s.Namespace": "payment-ns",
    "K8s.Pod": "payment-pod-123",
    "K8s.Node": "worker-node-1",
    "K8s.Workload": "payment-deployment",
    "RequestLatency": 135.5,
    "timestamp": 1622163600000
}
```

Generated Entity

The above EMF log will generate the following Entity:

```
{
    "KeyAttributes": {
        "Type": "Service",
        "Name": "PaymentService",
        "Environment": "Production"
    },
    "Attributes": {
        "PlatformType": "K8s",
        "K8s.Cluster": "main-cluster",
        "K8s.Namespace": "payment-ns",
```

```
        "K8s.Pod": "payment-pod-123",
        "K8s.Node": "worker-node-1",
        "K8s.Workload": "payment-deployment"
    }
}
```

Entity processing

CloudWatch processes the entity information as follows:

- **KeyAttributes:**
  - Determines entity type based on required fields
  - For Service-type, extracts Service name and Environment
  - These become the primary identifiers for the entity
- **Attributes:**
  - Sets PlatformType based on included platform attributes
  - Includes all relevant platform-specific information
  - Maintains relationship context for the telemetry data

CloudWatch uses this entity information to establish relationships between different pieces of telemetry data, enabling enhanced observability and contextual analysis of your applications and infrastructure. For more information, see How to add related information to custom telemetry sent to CloudWatch .

> **Note**
>
> Entity information helps CloudWatch create a complete picture of your application's telemetry data and its relationships within your infrastructure.

Using the PutLogEvents API to send manually-created embedded metric format logs

You can send embedded metric format logs to CloudWatch Logs using the CloudWatch Logs PutLogEvents API. When calling PutLogEvents, you have the option to include the following HTTP header, which tells CloudWatch Logs the metrics should be extracted, but it's not required.

```
x-amzn-logs-format: json/emf
```

The following is a full example using the AWS SDK for Java 2.x:

```
package org.example.basicapp;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatchlogs.CloudWatchLogsClient;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsRequest;
import software.amazon.awssdk.services.cloudwatchlogs.model.DescribeLogStreamsResponse;
import software.amazon.awssdk.services.cloudwatchlogs.model.InputLogEvent;
import software.amazon.awssdk.services.cloudwatchlogs.model.PutLogEventsRequest;

import java.util.Collections;

public class EmbeddedMetricsExample {
        public static void main(String[] args) {

                final String usage = "To run this example, supply a Region code (eg.
 us-east-1), log group, and stream name as command line arguments"
                                + "Ex: PutLogEvents <region-id> <log-group-name>
 <stream-name>";

                if (args.length != 3) {
                        System.out.println(usage);
                        System.exit(1);
                }

                String regionId = args[0];
                String logGroupName = args[1];
                String logStreamName = args[2];

                CloudWatchLogsClient logsClient =
 CloudWatchLogsClient.builder().region(Region.of(regionId)).build();

                // Build a JSON log using the EmbeddedMetricFormat.
                long timestamp = System.currentTimeMillis();
                String message = "{" +
                                "  \"_aws\": {" +
                                "    \"Timestamp\": " + timestamp  + "," +
                                "    \"CloudWatchMetrics\": [" +
                                "      {" +
                                "        \"Namespace\": \"MyApp\"," +
```

```
                                "       \"Dimensions\": [[\"Operation\"], [\"Operation
\", \"Cell\"]]," +
                                "       \"Metrics\": [{ \"Name\": \"ProcessingLatency
\", \"Unit\": \"Milliseconds\", \"StorageResolution\": 60 }]" +
                                "      }" +
                                "    ]" +
                                "  }," +
                                "  \"Operation\": \"Aggregator\"," +
                                "  \"Cell\": \"001\"," +
                                "  \"ProcessingLatency\": 100" +
                                "}";
                InputLogEvent inputLogEvent = InputLogEvent.builder()
                        .message(message)
                        .timestamp(timestamp)
                        .build();

                // Specify the request parameters.
                PutLogEventsRequest putLogEventsRequest = PutLogEventsRequest.builder()
                        .logEvents(Collections.singletonList(inputLogEvent))
                        .logGroupName(logGroupName)
                        .logStreamName(logStreamName)
                        .build();

                logsClient.putLogEvents(putLogEventsRequest);

                System.out.println("Successfully put CloudWatch log event");
        }

}
```

Note

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `AWS/Logs` namespace of your account. These can be used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Using the CloudWatch agent to send embedded metric format logs

This section describes how to install and use the CloudWatch agent. The first part of this section describes how to install the CloudWatch agent. The second part of this section describes how to

use the CloudWatch agent to send embedded metric format logs. If you want to use this method, you must install the CloudWatch agent for the AWS services you want to send embedded metric format logs from. Then you can begin sending the events. The CloudWatch agent must be version 1.230621.0 or later.

> **Note**
>
> You do not need to install the CloudWatch agent to send logs from Lambda functions. Lambda function timeouts are not handled automatically. This means that if your function times out before the metrics get flushed, then the metrics for that invocation will not be captured.

Installing the CloudWatch agent

Install the CloudWatch agent for each service which is to send embedded metric format logs.

Installing the CloudWatch agent on EC2

First, install the CloudWatch agent on the instance. For more information, see Installing the CloudWatch agent.

After you have installed the agent, configure the agent to listen on a UDP or TCP port for the embedded metric format logs. The following is an example of this configuration that listens on the default socket `tcp:25888`. For more information about agent configuration, see Manually create or edit the CloudWatch agent configuration file.

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Installing the CloudWatch agent on Amazon ECS

The easiest way to deploy the CloudWatch agent on Amazon ECS is to run it as a sidecar, defining it in the same task definition as your application.

Create agent configuration file

Create your CloudWatch agent configuration file locally. In this example, the relative file path will be `amazon-cloudwatch-agent.json`.

For more information about agent configuration, see Manually create or edit the CloudWatch agent configuration file.

```
{
  "logs": {
    "metrics_collected": {
      "emf": { }
    }
  }
}
```

Push configuration to SSM Parameter Store

Enter the following command to push the CloudWatch agent configuration file to the AWS Systems Manager (SSM) Parameter Store.

```
aws ssm put-parameter \
    --name "cwagentconfig" \
    --type "String" \
    --value "`cat amazon-cloudwatch-agent.json`" \
    --region "{{region}}"
```

Configure the task definition

Configure your task definition to use the CloudWatch Agent and expose the TCP or UDP port. The sample task definition that you should use depends on your networking mode.

Notice that the `webapp` specifies the `AWS_EMF_AGENT_ENDPOINT` environment variable. This is used by the library and should point to the endpoint that the agent is listening on. Additionally, the `cwagent` specifies the `CW_CONFIG_CONTENT` as a "valueFrom" parameter that points to the SSM configuration that you created in the previous step.

This section contains one example for bridge mode and one example for host or awsvpc mode. For more examples of how you can configure the CloudWatch agent on Amazon ECS, see the Github samples repository

The following is an example for bridge mode. When bridge mode networking is enabled, the agent needs to be linked to your application using the `links` parameter and must be addressed using the container name.

```
{
  "containerDefinitions": [
          {
              "name": "webapp",
              "links": [ "cwagent" ],
              "image": "my-org/web-app:latest",
              "memory": 256,
              "cpu": 256,
              "environment": [{
                "name": "AWS_EMF_AGENT_ENDPOINT",
                "value": "tcp://cwagent:25888"
              }],
          },
          {
              "name": "cwagent",
              "mountPoints": [],
              "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
              "essential": false,
              "memory": 256,
              "cpu": 256,
              "portMappings": [{
                  "protocol": "tcp",
                  "containerPort": 25888
              }],
              "environment": [{
                "name": "CW_CONFIG_CONTENT",
                "valueFrom": "cwagentconfig"
              }],
          }
      ],
}
```

The following is an example for host mode or awsvpc mode. When running on these network modes, the agent can be addressed over `localhost`.

```
{
  "containerDefinitions": [
          {
```

```
            "name": "webapp",
            "image": "my-org/web-app:latest",
            "memory": 256,
            "cpu": 256,
            "environment": [{
              "name": "AWS_EMF_AGENT_ENDPOINT",
              "value": "tcp://127.0.0.1:25888"
            }],
        },
        {
            "name": "cwagent",
            "mountPoints": [],
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "essential": false,
            "memory": 256,
            "cpu": 256,
            "portMappings": [{
                "protocol": "tcp",
                "containerPort": 25888
            }],
            "environment": [{
              "name": "CW_CONFIG_CONTENT",
              "valueFrom": "cwagentconfig"
            }],
        }
    ],
}
```

Note

In awsvpc mode, you must either give a public IP address to the VPC (Fargate only), set up a NAT gateway, or set up a CloudWatch Logs VPC endpoint. For more information about setting up a NAT, see NAT Gateways. For more information about setting up a CloudWatch Logs VPC endpoint, see Using CloudWatch Logs with Interface VPC Endpoints.
The following is an example of how to assign a public IP address to a task that uses the Fargate launch type.

```
aws ecs run-task \
--cluster {{cluster-name}} \
--task-definition cwagent-fargate \
--region {{region}} \
--launch-type FARGATE \
```

```
--network-configuration
 "awsvpcConfiguration={subnets=[{{subnetId}}],securityGroups=[{{sgId}}],assignPublicIp=ENA
```

Required permissions

Make sure the IAM role executing your tasks has permission to read from the SSM Parameter Store. You can add this permission by attaching the **AmazonSSMReadOnlyAccess** policy. To do so, enter the following command.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess
 \
--role-name CWAgentECSExecutionRole
```

Installing the CloudWatch agent on Amazon EKS

Parts of this process can be skipped if you have already installed CloudWatch Container Insights on this cluster.

Permissions

If you have not already installed Container Insights, then first make sure that your Amazon EKS nodes have the appropriate IAM permissions. They should have the **CloudWatchAgentServerPolicy** attached. For more information, see Setup guide (AWS CLI).

Create ConfigMap

Create a ConfigMap for the agent. The ConfigMap also tells the agent to listen on a TCP or UDP port. Use the following ConfigMap.

```
# cwagent-emf-configmap.yaml
apiVersion: v1
data:
  # Any changes here must not break the JSON format
  cwagentconfig.json: |
    {
      "agent": {
        "omit_hostname": true
      },
      "logs": {
        "metrics_collected": {
          "emf": { }
```

```
        }
      }
    }
kind: ConfigMap
metadata:
  name: cwagentemfconfig
  namespace: default
```

If you have already installed Container Insights, add the following `"emf": { }` line to your existing ConfigMap.

Apply the ConfigMap

Enter the following command to apply the ConfigMap.

```
kubectl apply -f cwagent-emf-configmap.yaml
```

Deploy the agent

To deploy the CloudWatch agent as a sidecar, add the agent to your pod definition, as in the following example.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
    # Your container definitions go here
    - name: web-app
      image: my-org/web-app:latest
    # CloudWatch Agent configuration
    - name: cloudwatch-agent
      image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
      imagePullPolicy: Always
      resources:
        limits:
          cpu: 200m
          memory: 100Mi
        requests:
          cpu: 200m
          memory: 100Mi
```

```
      volumeMounts:
        - name: cwagentconfig
          mountPath: /etc/cwagentconfig
      ports:
  # this should match the port configured in the ConfigMap
        - protocol: TCP
          hostPort: 25888
          containerPort: 25888
  volumes:
    - name: cwagentconfig
      configMap:
        name: cwagentemfconfig
```

Using the CloudWatch agent to send embedded metric format logs

When you have the CloudWatch agent installed and running, you can send the embedded metric format logs over TCP or UDP. There are two requirements when sending the logs over the agent:

- The logs must contain a `LogGroupName` key that tells the agent which log group to use.
- Each log event must be on a single line. In other words, a log event cannot contain the newline (\n) character.

The log events must also follow the embedded metric format specification. For more information, see Specification: Embedded metric format.

If you plan to create alarms on metrics created using embedded metric format, see Setting alarms on metrics created with the embedded metric format for recommendations.

The following is an example of sending log events manually from a Linux bash shell. You can instead use the UDP socket interfaces provided by your programming language of choice.

```
echo '{"_aws":{"Timestamp":1574109732004,"LogGroupName":"Foo","CloudWatchMetrics":
[{"Namespace":"MyApp","Dimensions":[["Operation"]],"Metrics":
[{"Name":"ProcessingLatency","Unit":"Milliseconds","StorageResolution":60}]}]},"Operation":"Agg
 \
> /dev/udp/0.0.0.0/25888
```

Note

With the embedded metric format, you can track the processing of your EMF logs by metrics that are published in the `AWS/Logs` namespace of your account. These can be

used to track failed metric generation from EMF, as well as whether failures happen due to parsing or validation. For more details see Monitoring with CloudWatch metrics.

Using the embedded metric format with AWS Distro for OpenTelemetry

OpenTelemetry is an open-source initiative that removes boundaries and restrictions between vendor-specific formats for tracing, logs, and metrics by offering a single set of specifications and APIs. For more information, see OpenTelemetry. You can use the embedded metric format as a part of the OpenTelemetry project. Using the embedded metric format with OpenTelemetry requires two components: an OpenTelemetry-compliant data source and the AWS Distro for OpenTelemetry Collector enabled for use with CloudWatch embedded metric format logs.

We have preconfigured redistributions of the OpenTelemetry components, which AWS maintains, to make onboarding as easy as possible. For more information about using OpenTelemetry with the embedded metric format, in addition to other AWS services, see AWS Distro for OpenTelemetry. For additional information regarding language support and usage, see AWS Observability on Github.

Viewing your metrics and logs in the console

After you generate embedded metric format logs that extract metrics, you can use the CloudWatch console to view the metrics. Embedded metrics have the dimensions that you specified when you generated the logs. Also, the embedded metrics that you generated using the client libraries have the following default dimensions:

- ServiceType
- ServiceName
- LogGroup

This section describes how to view these metrics in the CloudWatch console and query your extracted metrics using CloudWatch Logs Insights.

To view metrics that were generated from embedded metric format logs

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.

3. Select a namespace that you specified for your embedded metrics when you generated them. If you used the client libraries to generate the metrics and did not specify a namespace, then select **aws-embedded-metrics**. This is the default namespace for embedded metrics generated using the client libraries.
4. Select a metric dimension (for example, **ServiceName**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. You can do the following:

    a. To sort the table, use the column heading.

    b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.

    c. To filter by resource, choose the resource ID and then choose **Add to search**.

    d. To filter by metric, choose the metric name and then choose **Add to search**.

Querying logs using CloudWatch Logs Insights

You can query the detailed log events associated with the extracted metrics by using CloudWatch Logs Insights to provide deep insights into the root causes of operational events. One of the benefits of extracting metrics from your logs is that you can filter your logs later by the unique metric (metric name plus unique dimension set) and metric values, to get context on the events that contributed to the aggregated metric value

For example, to get an impacted request id or x-ray trace id, you could run the following query in CloudWatch Logs Insights.

```
filter Latency > 1000 and Operation = "Aggregator"
| fields RequestId, TraceId
```

You can also perform query-time aggregation on high-cardinality keys, such as finding the customers impacted by an event. The following example illustrates this.

```
filter Latency > 1000 and Operation = "Aggregator"
| stats count() by CustomerId
```

For more information, see Analyzing Log Data with CloudWatch Logs Insights

Setting alarms on metrics created with the embedded metric format

In general, creating alarms on metrics generated by the embedded metric format follows the same pattern as creating alarms on any other metrics. For more information, see Using Amazon CloudWatch alarms. Embedded metric format metric generation depends on your log publishing flow because CloudWatch Logs needs to process logs so they can be transformed into metrics. It's important to publish logs in a timely manner, so that metric datapoints are created within the period of time that alarms are evaluated.

If you plan on using the embedded metric format to send high-resolution metrics and create alarms on these metrics, we recommended that you flush logs to CloudWatch Logs at an interval of 5 seconds or less to avoid introducing an additional delay, which can cause alarming on partial or missing data. If you are using the CloudWatch agent, you can adjust the flush interval by setting the `force_flush_interval` parameter in the CloudWatch agent configuration file. This value defaults to 5 seconds. If you are using Lambda on other platforms where you can't control the log flush interval, consider using "M out of N" alarms to control the number of datapoints used to alarm. For more information, see Alarm evaluation.

CloudWatch pipelines

CloudWatch pipelines is a fully managed data collector that ingests, transforms, and routes telemetry data—including logs and metrics—from AWS services, third-party applications, and custom sources to CloudWatch. Built-in processors let you enrich, filter, and standardize logs into formats like OCSF—without managing infrastructure or third-party tools. For metrics, CloudWatch pipelines processes and enriches OpenTelemetry (OTel) metrics inline during ingestion into CloudWatch.

CloudWatch pipelines is fully integrated with the logs management experience, enabling you to consistently process and enrich log data across related log groups through data-source and data-type specification. This unlocks use cases such as:

- **Automatic log categorization** – Logs processed through pipelines are automatically tagged with data source information, enabling service-centric discovery and querying across your infrastructure
- **Expanding third-party support** – Aggregate and normalize logs from a growing library of third-party sources for unified analytics and compliance

For metrics, CloudWatch pipelines enables the following use cases:

- **Enrichment** – Add business context such as team, cost center, or environment to metrics from sources you cannot modify
- **Cost optimization** – Strip high-cardinality attributes to reduce storage costs
- **Standardization** – Rename metrics and attributes to enforce naming conventions

Output from pipelines is fully compatible with CloudWatch Logs features including Logs Insights queries, Logs Anomaly Detection, and Live Tail. CloudWatch pipelines works with both Standard and Infrequent Access log classes and is backwards compatible with Log Transformers.

For metrics pipelines, processed metrics are fully queryable through PromQL in Query Studio and compatible with CloudWatch Alarms and Anomaly Detection.

To get started with CloudWatch pipelines, visit pipelines within the CloudWatch ingestion page in the CloudWatch console.

> **Note**
>
> Be aware of the following limits that apply to CloudWatch pipelines
>
> - Logs pipelines per account: 330
> - Up to 300 pipelines for collecting data from CloudWatch Logs
> - Up to 30 pipelines for collecting data from other sources
> - Metrics pipelines per account: up to 300
> - Up to 20 processors per pipeline (logs and metrics)
> - Metrics pipelines supported ingestion path: OTLP only

Pipeline components

Each pipeline consists of the following components:

- **Source** – Defines where data originates from (Amazon S3 buckets, CloudWatch Logs, CloudWatch Metrics (OTel), third-party integration). Each pipeline must have exactly one source. Metrics pipelines process OTel metrics on the OTLP ingestion path.

- **Processors** (optional) – Transform, parse, and enrich telemetry data as it flows through the pipeline. Processors are applied sequentially in the order they are defined. Metrics pipelines support a distinct set of processors optimized for OTel metric attributes.
- **Sink** – Defines the destination where processed data is sent. Supported sink types include `cloudwatch_logs` and `cloudwatch_metrics`. A logs pipeline can use up to 10 ordered `cloudwatch_logs` sinks for conditional routing. A metrics pipeline must have exactly one sink.
- **Extensions** (optional) – Provide additional functionality such as AWS Secrets Manager integration for credential management.

Throughout the entire pipeline, your data remains protected with transport layer encryption so that your data meets security and compliance requirements.

> **Note**
>
> Pipeline definitions are not encrypted with customer-provided keys and must never include sensitive data, such as passwords, API keys, or personally identifiable information (PII).

> **Note**
>
> Adding processors leads to mutation of the log events and original (raw) logs are not retained.

Pricing

CloudWatch pipelines is included with CloudWatch Logs at no additional cost. Standard log ingestion rates based on log class (vended or custom) and storage class (Standard or Infrequent Access) still apply. Metering occurs at time of first ingestion into CloudWatch. CloudWatch Logs sources are metered before pipeline processing. Third-party and S3 bucket sources are classified as Custom logs and metered after processing. For pricing details, see CloudWatch Pricing.

For metrics pipelines, standard CloudWatch OTel metrics ingestion and storage charges apply. There is no additional charge for pipeline processing.

Region availability

CloudWatch pipelines is available in the following AWS Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Asia Pacific (Malaysia)
- Asia Pacific (Melbourne)
- Asia Pacific (Mumbai)
- Asia Pacific (New Zealand)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Taipei)
- Asia Pacific (Thailand)
- Asia Pacific (Tokyo)
- Canada (Central)
- Canada West (Calgary)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)

- Europe (Zurich)
- AWS European Sovereign Cloud (Germany)
- Israel (Tel Aviv)
- Mexico (Central)
- South America (São Paulo)

> **Note**
>
> Third-party data source collection is available in regions where OpenSearch Ingestion has API endpoints.

For more details, see Amazon CloudWatch endpoints and quotas in the *AWS General Reference*.

Creating pipelines

CloudWatch pipelines supports creating pipelines for both logs and metrics data. The following sections describe how to create each type of pipeline.

Creating a logs pipeline

The pipeline configuration wizard guides you through creating your logs data pipeline.

1. Under **General settings**, provide the data source details including source name and type. You can also specify pipeline tags and the name of your pipeline.
2. Under **Destination**, specify the destination details. For logs pipelines, CloudWatch Logs is the only supported destination. For metrics pipelines, CloudWatch Metrics is the only supported destination.
3. Under **Processor**, add the desired processors and parsers. A parser is a required first step for certain data types. You can perform custom parsing using processors like Grok or CSV. You can use the lookup processor to enrich log events with data from an Amazon CloudWatch Logs lookup table. Processors that are not supported by the data type are disabled.

   To configure processors using natural language, enable the **AI-assisted** toggle. Enter a description of the log transformations you need, and CloudWatch pipelines generates the processor configuration automatically. For AWS vended logs, a sample log event is also

generated so you can verify the output before deploying. You can review and edit the generated configuration before creating the pipeline.

> **Important**
>
> To use AI-assisted processor configuration, you must have the `logs:GeneratePipeline` IAM permission. For more information, see the section called "AI-assisted processor configuration permissions".

You can also add conditional processing rules to supported processors using the `when` parameter. Conditional processing lets you control which log entries a processor acts on. For the expression syntax and supported processors, see the section called "Expression syntax for conditional processing".

To preserve unmodified copies of your log data for audit or compliance purposes, enable the **Keep original log** toggle. When enabled, CloudWatch pipelines automatically stores a copy of each raw log event before any transformation takes place in the `@original_message` system field of the event. This ensures that original data is always available for audits or investigations, even after processors modify the log events. The **Keep original log** toggle is only available for pipelines with a CloudWatch vended log source.

4. Under **Review and create**, review the pipeline configuration. If you're satisfied with the configuration, choose **Create pipeline** to start deployment and creation of pipeline resources. Pipeline creation completion takes up to 5 minutes depending on the source type. Upon completion, you'll be taken to the Pipelines tab in the Ingestion Console.

> **Important**
>
> Pipeline processor configurations are logged in AWS CloudTrail events for auditing and compliance purposes. To protect sensitive information, do not include passwords, API keys, or other sensitive information in processor configurations.

Creating a metrics pipeline

You can create a metrics pipeline by using the CloudWatch console or the AWS CLI. With a metrics pipeline, you can process CloudWatch metrics data, apply transformations, and forward the processed metrics to CloudWatch.

To create a metrics pipeline (console)

Use the following procedure to create a metrics pipeline in the CloudWatch console.

1. Open the CloudWatch console and in the navigation pane, choose **Ingestion**, **Pipelines**.
2. Choose **Create pipeline**.
3. For the source type, select **CloudWatch Metrics (OTel)**.
4. Configure the selection criteria to specify which metrics the pipeline processes. Selection criteria use OTel attribute paths such as `resource.attributes`, `metric.name`, and `datapoint.attributes`. All criteria use AND semantics (all must match).
5. Under **Processor**, add up to 20 processors to transform your metrics data.
6. Under **Review and create**, review the pipeline configuration and choose **Create pipeline**.

To create a metrics pipeline (AWS CLI)

Use the `create-telemetry-pipeline` command to create a metrics pipeline from the AWS CLI.

```
aws observabilityadmin create-telemetry-pipeline \
    --pipeline-configuration-body file://config.yaml
```

The following example shows a YAML configuration file that defines a metrics pipeline with a source, processor, and sink.

```
pipeline:
  source:
    cloudwatch_metrics:
      format: otlp
      selection_criteria:
        - match_all:
            - 'resource.attributes["service.name"] == "payment-service"'
            - 'metric.name == "http.server.request.duration"'
  processor:
```

```
    - add_attributes:
        attributes:
          - key: resource.attributes["team"]
            value: "platform-engineering"
  sink:
    - cloudwatch_metrics: {}
```

When the pipeline is created, it returns an ARN in the following format:

```
arn:aws:observabilityadmin:region:accountId:telemetry-pipeline/pipelineID
```

> **Note**
>
> CloudWatch Metrics is the only supported destination for metrics pipelines.

Supported API operations

The following table lists the API operations that you can use to manage metrics pipelines.

Operation	Description
`CreateTelemetryPipeline`	Creates a new metrics pipeline.
`GetTelemetryPipeline`	Retrieves details about a specific metrics pipeline.
`ListTelemetryPipelines`	Lists all metrics pipelines in your account.
`DeleteTelemetryPipeline`	Deletes a metrics pipeline.
`ValidateTelemetryPipelineConfiguration`	Validates a pipeline configuration before creation.
`UpdateTelemetryPipeline`	Updates an existing metrics pipeline configuration.

Managing pipelines

After creating pipelines, you can monitor their performance and manage their configuration through the Pipelines tab.

Pipeline status monitoring

Each pipeline displays real-time status information including:

- Pipeline status: Active, Creating, Updating, Deleting, Create Failed, Update Failed, or Delete Failed.
- Data throughput metrics
- Error rates and failure details

Pipeline operations

You can perform the following operations on existing pipelines:

- **View details** – Review pipeline configuration and status
- **Edit pipeline** – Edit pipeline configuration including processors, parsing, and source selection criteria
- **Delete pipeline** – Remove pipelines that are no longer needed

> **Note**
>
> To programmatically manage metrics pipelines, use the Observability Admin APIs including `CreateTelemetryPipeline`, `UpdateTelemetryPipeline`, and `DeleteTelemetryPipeline`.

Supported data sources for CloudWatch

CloudWatch collects and processes telemetry data from a wide range of sources to provide unified observability and security insights. Data sources fall into four categories: AWS native services, third-party platforms, custom sources, and CloudWatch Metrics (OTel).

You can automate the enablement of AWS data sources using telemetry configuration. For more information, see Telemetry discovery and enablement.

AWS service data sources

CloudWatch provides native integration with 90+ AWS services for automatic data collection. When an AWS service source is selected, CloudWatch pipelines intercepts logs ingested into CloudWatch Logs for processing. To get started, enable logging for the supported AWS services using the service's console, then select the data source and type in the CloudWatch pipelines creation wizard.

The following table highlights key AWS service data sources. For the complete list of 90+ supported services, see Supported AWS services for data sources.

AWS service	Data type	Description
Amazon Amazon VPC	Flow Logs	Network traffic metadata for Amazon VPCs, subnets, and network interfaces
Amazon Amazon EKS	Control Plane Logs	Kubernetes API server, audit, authenticator, controller manager, and scheduler logs
AWS WAF	Web ACL Logs	Web request inspection logs including rule match details and actions taken
Amazon Route 53	Resolver Query Logs	DNS query logs for Amazon VPC resources routed through Route 53 Resolver
CloudTrail	Management and Data Events	API activity and resource-level operations across AWS services
Amazon Amazon EC2	Detailed Metrics	Instance-level performance metrics at 1-minute granularity
AWS Security Hub	CSPM Findings	Cloud security posture managemen t findings from AWS and third-party providers

AWS service	Data type	Description
Amazon Bedrock AgentCore	Runtime, Browser, CodeInterpreter, Gateway, Memory	Agent runtime execution, browser interaction, code execution, gateway, and memory operation logs
Amazon CloudFront	Distribution Logs	CDN access logs for content delivery distributions
Network Load Balancer	Access Logs	Network Load Balancer connection and TLS negotiation logs
Amazon RDS	Aurora MySQL and Aurora PostgreSQL Logs	Audit, error, general, slow query, IAM DB authentication error, proxy, and enhanced monitoring logs

For more information about CloudWatch Logs data sources, see Data source discovery and management.

Third-party data sources

CloudWatch extends monitoring capabilities beyond AWS with direct integrations for 38 third-party security, identity, and endpoint platforms. These integrations consolidate security events, audit logs, and telemetry data from external sources into CloudWatch Logs for unified analysis.

The following table lists the supported direct third-party integrations:

Source	Integration pattern	Category
Akamai DataStream 2	S3 Delivery	CDN and edge security
Check Point NGFW	S3 Delivery	Network security
Cisco Duo	API	Identity and access management
Cisco Meraki	API	Network security
Cisco Umbrella	S3 Delivery	DNS and network security

Source	Integration pattern	Category
CrowdStrike Falcon	S3 Delivery	Endpoint security
Drupal Core	API	Content management
Entrust IDaaS	API	Identity and access management
F5 BIG-IP	S3 Delivery	Network security
GitHub	API	Source code and audit logs
GitLab	API	DevSecOps and source code
HashiCorp Vault	S3 Delivery	Secrets management
Jamf Protect	S3 Delivery	Endpoint security
Microsoft Entra ID	API	Identity and access management
Microsoft Office 365	API	Productivity and audit logs
Microsoft Windows Event Logs	API	Operating system events
Netskope	API	Network security and CASB
Okta Auth0	API	Identity and access management
Okta SSO	API	Identity and access management
OneLogin Identity	API	Identity and access management
Palo Alto Networks NGFW	API	Network security
Palo Alto Prisma Cloud	API	Cloud security
PingIdentity PingAccess	S3 Delivery	Access management
PingIdentity PingFederate	S3 Delivery	Identity federation
PingIdentity PingOne	API	Identity and access management

Source	Integration pattern	Category
Proofpoint TAP	API	Email security
Slack Audit Log	API	Collaboration and audit logs
SentinelOne	S3 Delivery	Endpoint security
ServiceNow CMDB	API	IT service management
Tanium Endpoint Management	S3 Delivery	Endpoint security and management
Tenable Vulnerability Management	API	Vulnerability management
Wazuh Platform	API	Security monitoring and threat detection
Box	API	Content management and file sharing
Jamf Pro	API	Device management (Apple)
Broadcom Carbon Black	S3 Delivery	Endpoint detection and response
Wiz CNAPP	API	Cloud security
Zeek	S3 Delivery	Network security monitoring
Zscaler ZIA/ZPA	S3 Delivery	Network security

For detailed setup procedures, prerequisites, and configuration steps for each integration, see Third-party data sources integration.

Additional third-party sources through Security Hub CSPM

Beyond the 34 direct integrations, 49+ additional third-party sources are available through AWS Security Hub CSPM integration. Security Hub CSPM partner providers that send

findings to Security Hub are automatically available as data sources. For the full list of supported partners, see the Security Hub CSPM partner providers documentation.

Additional third-party sources through Security Hub

AWS Security Hub (distinct from Security Hub CSPM) provides its own set of third-party integrations. These integrations that send findings to Security Hub are automatically available as data sources. For the full list of supported integrations, see the Security Hub third-party integrations documentation.

Custom data sources

For logs that are not covered by AWS service or third-party integrations, CloudWatch pipelines can process custom logs stored in CloudWatch Logs or Amazon S3 buckets. Custom sources accommodate unique organizational requirements:

- **Syslog data** – Messages ingested through the CloudWatch Logs Syslog service
- **Application-specific logs** – Custom application telemetry from Amazon EC2 instances with specialized logging formats
- **File-based ingestion** – Amazon S3-based log files from legacy systems or batch processing workflows
- **Serverless integration** – Lambda function logs and custom serverless application telemetry

For more details, see Custom log data from CloudWatch Logs or an Amazon S3 bucket.

CloudWatch Metrics (OTel) source

The CloudWatch Metrics (OTel) source processes OpenTelemetry (OTel) metrics on the OTLP ingestion path based on selection criteria. This source enables you to transform metrics data through CloudWatch pipelines before the metrics are persisted in CloudWatch Metrics.

The following metric types are supported:

- **Custom OTel metrics** – Metrics that your applications emit through OpenTelemetry instrumentation and send to the OTLP endpoint
- **AWS vended OTel metrics** – Metrics that AWS services emit through OTLP

> **Note**
>
> The CloudWatch Metrics (OTel) source only supports metrics ingested through the OTLP endpoint. Metrics sent through other ingestion paths, such as the `PutMetricData` API, are not processed by this source.

You define which metrics to process by configuring metrics pipeline selection criteria. Selection criteria let you target specific metrics based on OTel attributes such as resource attributes, metric name, and datapoint attributes. For more information, see the section called “CloudWatch Metrics (OTel)”.

For information about sending OTel metrics to CloudWatch, see Using OpenTelemetry with CloudWatch.

Topics

- CloudWatch Metrics (OTel)
- Third-party data sources integration
- Custom log data from CloudWatch Logs or an Amazon S3 bucket
- AWS service logs from CloudWatch Logs

CloudWatch Metrics (OTel)

Selection criteria determine which OTel metrics enter a pipeline for processing. Each criterion is an expression of the form `<path> == "<value>"` that matches against a specific attribute of the incoming metric. At least one selection criterion is required.

Criteria are grouped inside a `match_all` block with AND semantics — a metric must match every expression in the group to enter the pipeline.

Supported paths

The following OTTL paths are supported in selection criteria:

Path	Description
`resource.attributes["key"]`	Resource-level attribute

Path	Description
`instrumentation_scope.name`	Instrumentation scope name
`instrumentation_scope.version`	Instrumentation scope version
`instrumentation_scope.attributes["key"]`	Instrumentation scope attribute
`metric.name`	Metric name
`datapoint.attributes["key"]`	Datapoint-level attribute
`attributes["key"]`	Short form for `datapoint.attributes["key"]`

Configuration

Selection criteria are defined in the `source` section of the pipeline configuration:

```
pipeline:
  source:
    cloudwatch_metrics:
      format: otlp
      selection_criteria:
        - match_all:
            - 'resource.attributes["service.name"] == "my-service"'
            - 'metric.name == "http.server.request.duration"'
  processor:
    - add_attributes:
        attributes:
          - key: resource.attributes["team"]
            value: "platform-engineering"
  sink:
    - cloudwatch_metrics: {}
```

The following example uses all supported path types:

```
pipeline:
  source:
    cloudwatch_metrics:
```

```
      format: otlp
      selection_criteria:
        - match_all:
            - 'resource.attributes["service.name"] == "my-service"'
            - 'instrumentation_scope.name == "my-scope"'
            - 'instrumentation_scope.version == "1.0.0"'
            - 'instrumentation_scope.attributes["library"] == "otel-java"'
            - 'metric.name == "http.server.request.duration"'
            - 'datapoint.attributes["status_code"] == "200"'
            - 'attributes["environment"] == "production"'
  processor:
    - add_attributes:
        attributes:
          - key: resource.attributes["team"]
            value: "observability"
  sink:
    - cloudwatch_metrics: {}
```

Verifying with PromQL

CloudWatch maps OTLP attribute scopes to PromQL labels using the @ prefix convention. Use this mapping to verify that your pipeline is processing metrics as expected in Query Studio:

Pipeline OTTL path	PromQL label prefix	Example
`resource.attributes["key"]`	`@resource.`	`@resource.service.name`
`instrumentation_scope.name`	`@instrumentation.@name`	`@instrumentation.@name`
`instrumentation_scope.attributes["key"]`	`@instrumentation.`	`@instrumentation.library`
`datapoint.attributes["key"]` / `attributes["key"]`	`@datapoint.` or bare	`status_code`

For example, if your pipeline adds `resource.attributes["team"]` with value `"platform-engineering"`, you can confirm it was applied:

```
{"CPUUtilization", "@resource.team"="platform-engineering"}
```

Requirements and limitations

Single `match_all` group

Each pipeline supports exactly one `match_all` group in `selection_criteria`. You cannot define multiple `match_all` groups in a single pipeline.

Minimum criteria

At least one expression is required in the `match_all` group.

Maximum criteria

A `match_all` group can contain at most 20 expressions.

AND semantics

All expressions in the `match_all` group must match for a metric to enter the pipeline.

Exact string matching

Values must be static strings. Wildcards, regular expressions, and partial matches are not supported. Each expression must use the == operator.

No overlapping criteria across pipelines

Each metric datapoint must match at most one pipeline. If a datapoint matches both new and existing pipeline criteria, pipeline creation fails. To prevent overlap, include at least one attribute path with distinct values in each pipeline's selection criteria.

For example, the following two selection criteria overlap because Pipeline A selects all metrics from `payment-service`, which includes the specific metric that Pipeline B targets:

```
# Pipeline A — selects ALL metrics from payment-service
selection_criteria:
  - match_all:
      - 'resource.attributes["service.name"] == "payment-service"'

# Pipeline B — FAILS: a datapoint with service.name="payment-service"
# and metric.name="http.server.request.duration" matches both pipelines
selection_criteria:
```

```
  - match_all:
      - 'metric.name == "http.server.request.duration"'
```

Third-party data sources integration

Integrating CloudWatch pipelines with your third-party data source let you connect external security tools, identity providers, and monitoring platforms to CloudWatch pipelines for centralized data analysis. This integration consolidates security events, audit logs, and telemetry data from multiple sources.

> **Note**
>
> Data collected from third-party sources is mutated to adhere to the required schema when it is collected by CloudWatch pipelines. The original data source is not retained by CloudWatch.

Third-party data can be collected using two methods:

1. **Direct API Integration** – Some sources offer Event Stream APIs where you only need to provide API credentials to configure the connector
2. **S3 Bucket Integration** – Data from sources can be ingested into a customer-managed S3 bucket for CloudWatch pipelines to collect

The following table identified the integrations methods used by the supported third-party data platforms:

Source	Integration pattern	Requires S3 bucket	Requires SQS queue	Uses Secrets Manager extension	Required IAM policies
Akamai DataStream 2	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Check Point NGFW	S3 Delivery	Yes	Yes	No	Source-specific IAM policies

Source	Integration pattern	Requires S3 bucket	Requires SQS queue	Uses Secrets Manager extension	Required IAM policies
Cisco Duo	API	No	No	Yes	API caller permissions
Cisco Meraki	API	No	No	Yes	API caller permissions
Cisco Umbrella	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
CrowdStrike Falcon	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Drupal Core	API	No	No	Yes	API caller permissions
Entrust IDaaS	API	No	No	Yes	API caller permissions
F5 BIG-IP	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
GitHub	API	No	No	Yes (optional)	API caller permissions
GitLab	API	No	No	Yes	API caller permissions
HashiCorp Vault	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Jamf Protect	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Microsoft Entra ID	API	No	No	Yes	API caller permissions

Source	Integration pattern	Requires S3 bucket	Requires SQS queue	Uses Secrets Manager extension	Required IAM policies
Microsoft Office 365	API	No	No	Yes	API caller permissions
Microsoft Windows Event Logs	API	No	No	Yes	API caller permissions
Netskope	API	No	No	Yes	API caller permissions
Okta Auth0	API	No	No	Yes	API caller permissions
Okta SSO	API	No	No	Yes	API caller permissions
OneLogin Identity	API	No	No	Yes	API caller permissions
Palo Alto Networks NGFW	API	No	No	Yes	API caller permissions
Palo Alto Prisma Cloud	API	No	No	Yes	API caller permissions
PingIdentity PingAccess	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
PingIdentity PingFederate	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
PingIdentity PingOne	API	No	No	Yes	API caller permissions

Source	Integration pattern	Requires S3 bucket	Requires SQS queue	Uses Secrets Manager extension	Required IAM policies
Proofpoint TAP	API	No	No	Yes	API caller permissions
Slack Audit Log	API	No	No	Yes	API caller permissions
SentinelOne	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
ServiceNow CMDB	API	No	No	Yes	API caller permissions
Tanium Endpoint Management	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Tenable Vulnerability Management	API	No	No	Yes	API caller permissions
Wiz CNAPP	API	No	No	Yes	API caller permissions
Zeek	S3 Delivery	Yes	Yes	No	Source-specific IAM policies
Zscaler ZIA/ZPA	S3 Delivery	Yes	Yes	No	Source-specific IAM policies

Third-party integration sent through Security Hub CSPM is also supported. For comprehensive information about Security Hub's third-party integrations, including supported partners and integration configurations with "Sends findings" direction, refer to the Security Hub Third-Party Integration documentation.

AWS Security Hub (distinct from Security Hub CSPM) also supports third-party integrations as data sources. For the full list of supported integrations, refer to the Security Hub third-party integrations documentation.

Data transformation and standardization

Third-party integrations support data transformation to standardized formats for consistent analysis:

- **Open Cybersecurity Schema Framework (OCSF)** – Converts security events from different vendors into a common schema for unified threat detection and analysis. Because OCSF is only for certain event classes, not all raw events are mapped to OCSF.
- **Custom transformations** – Pipeline processors that normalize data formats, enrich events with additional context, and filter relevant information.
- **Field mapping** – Automatic mapping of vendor-specific fields to standardized field names for consistent querying and analysis.

> **Note**
>
> Storing telemetry data from third-party sources in OCSF is an optional feature that might not be available for all data sources.

Log group

Third-party data is ingested into a CloudWatch log group. If you are using the AWS Management Console to configure CloudWatch pipelines if the log group does not exist, it is created automatically through the wizard process.

Authentication and security

Third-party integrations use secure authentication methods to protect data in transit:

- **OAuth 2.0 and application registration** – Secure token-based authentication for cloud platforms like Microsoft and Okta.
- **API keys and certificates** – Encrypted authentication credentials for direct API access.
- **IAM roles and policies** – AWS Identity and Access Management integration for secure S3 bucket access and cross-account data sharing.

> **Note**
>
> Data collected from third-party sources is mutated to adhere to the required schema when it is collected by CloudWatch pipelines. The original data source is not retained by CloudWatch.

Each integration requires platform-specific configuration to establish secure data delivery to your AWS environment.

The following sections provide detailed setup procedures for supported third-party integrations. Each integration includes prerequisites, configuration steps, and validation procedures to make sure proper data flow.

Akamai DataStream 2 integration configuration

Akamai DataStream 2 is a near real-time log streaming service that delivers detailed performance and security metrics from the Akamai edge platform to various third-party destinations. It enables proactive monitoring, troubleshooting, and, through its support for DataStream 2.1, includes Edge DNS, Global Traffic Management (GTM), and EdgeWorkers log data.

Topics

- Source configuration for Akamai DataStream 2
- CloudWatch pipelines configuration for Akamai DataStream 2

Source configuration for Akamai DataStream 2

Integrating with Akamai DataStream 2

To integrate Akamai DataStream 2 Data Replicator with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your Akamai DataStream 2 source by configuring Amazon S3 and Amazon SQS to receive data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Log forwarding setup

Akamai DataStream 2 supports real-time log delivery through DataStream 2 to forward logs to Amazon S3.

Instructions to setup Amazon S3 and Amazon SQS

Configuring Akamai DataStream 2 to send logs to an Amazon S3 bucket involves several steps. These steps focus on setting up the Amazon S3 bucket, Amazon SQS queue, and IAM roles, and then configuring the CloudWatch pipeline.

- Create an Amazon S3 bucket that stores Akamai logs and create separate folders for each log type (for example, DNS, CDN, EdgeWorkers). Create an IAM user and grant S3 write permission, then create an access key and secret key for this account.
- Make sure the Akamai DataStream 2 log exporter is configured to send logs to Amazon S3. DataStream 2 uploads logs to Amazon S3 over TLS, so you need to enable server-side encryption for Amazon S3. Refer to the Akamai S3 streaming setup guide for detailed instructions.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Akamai DataStream 2, choose Akamai DataStream 2 as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Akamai DataStream 2 events that map to DNS Activity (4003), HTTP Activity (4002), and Base Event (0). Any events that don't conform to the mentioned OCSF schema are passed through.

DNS Activity (4003) contains the following event logs:

- DNS Logs
- GTM Logs

HTTP Activity (4002) contains the following event logs:

- CDN Logs

Base Event (0) contains the following event logs:

- EdgeWorkers Logs

> **Note**
>
> DNS and Global Traffic Management (GTM) logs share the same format. No separate OCSF mappings are required.

Events that do not match any OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

CloudWatch pipelines configuration for Akamai DataStream 2

The Akamai DataStream 2 setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Akamai DataStream 2 source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "akamai_datastream_2"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "akamai_datastream_2".

`aws.region` (required)

The AWS Region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

Box integration configuration

Box is a cloud content management and collaboration platform that enables secure file sharing, storage, and workflow automation across organizations. With CloudWatch Pipelines, you can use the Box REST API to retrieve enterprise events, user inventories, group structures, group memberships, and group collaborations from your Box enterprise. The pipeline authenticates using OAuth2 Client Credentials Grant and polls the API at configurable intervals to collect security and administrative data for monitoring and analysis.

Topics

- Source configuration for Box
- CloudWatch pipelines configuration for Box

Source configuration for Box

Prerequisites

Before you begin, make sure you have the following:

- A Box Enterprise account with **admin** privileges. **Two-factor authentication (2FA)** must be enabled.
- A Box App configured with OAuth2 **Client Credentials Grant** (Server Authentication) in the Box Developer Console.
- A Box App authorized for authentication in the Box Admin Console
- An AWS account with permissions to create and manage CloudWatch Pipelines.
- An AWS account with required permissions to create, retrieve, and update secrets in AWS Secrets Manager.
- An AWS account with permissions to create and manage CloudWatch Logs log groups.

Integrating with Box

To integrate Box with CloudWatch Pipelines, complete the following high-level steps:

- Create and configure a Custom App with Client Credentials Grant in the Box Developer Console.
- Verify 2FA and authorize the Custom App.
- Note the Client ID, Client Secret, and Enterprise ID.
- Store the client credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with Box as the data source.
- Verify data is flowing into the configured CloudWatch Logs log group.

Authenticating with Box

To read the data, the pipeline needs to authenticate with your Box enterprise. Box uses OAuth2 Client Credentials Grant, which issues a short-lived access token (60 minutes) for server-to-server API access without user interaction.

Configure Authentication for Box

To configure authentication credentials for the pipeline:

1. Log in to the Box Developer Console and create a **New App**.
2. Enter the App Name, choose **Client Credentials Grant** as the App Type, and choose **Create App**.
3. Navigate to **App Details > Access** panel, choose **Fetch Secret** and complete the 2FA verification.
4. Choose **Fetch Secret** again, and then copy the `Client ID` and `Client Secret` for later use.
5. Under App Access Level, select **App + Enterprise Access**. Keep the default selections for Application Scopes.
6. Navigate to **App Details > Status**, choose **Authorize**, and make sure the application is marked as Authorized.
7. Choose your profile icon and select **Copy Enterprise ID** for later use. Alternatively, you can find it in **Admin Console** > **Account & Billing** > **Enterprise ID**.
8. In AWS Secrets Manager, create a secret and store:
   - Client ID under the key `client_id`
   - Client Secret under the key `client_secret`

Configuring the CloudWatch Pipeline

To configure the pipeline to read data, choose Box as the data source. Provide the required information such as the Enterprise ID (`enterprise_id`), Client ID (`client_id`), and Client Secret (`client_secret`). After you create and activate the pipeline, enterprise events and inventory data from Box will begin flowing into the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Box events that map to the following OCSF event classes.

> **Note**
>
> Events that do not match any OCSF event classes listed below are automatically passed through and sent directly to the configured sink in their original format without additional processing.

Entity Management (3004)

Enterprise Events – Contains enterprise-wide administrative and security events including file operations, sharing changes, user management actions, Shield alerts, and application events.

Account Change (3001)

Users – Provides user account inventory including profile information, status, storage usage, and role assignments.

Group Management (3006)

- **Groups** – Contains group inventory with group type, membership visibility, and invitability settings.
- **Group Memberships** – Records user-to-group membership associations including role assignments (member, admin).
- **Group Collaborations** – Provides shared access grants (collaborations) between groups and content items including role and status.

Base Event (0)

Events that do not map to a specific OCSF class are mapped to the Base Event class.

CloudWatch pipelines configuration for Box

Collects enterprise events, user inventory, group inventory, group memberships, and group collaborations from Box using OAuth2 Client Credentials authentication.

Configure the Box source with the following parameters:

```
source:
  box:
    enterprise_id: "123456789"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:my-box-secret:client_id}}"
        client_secret: "${{aws_secrets:my-box-secret:client_secret}}"
```

Parameters

`enterprise_id` (required)

The Box Enterprise ID (numeric). Found in the Box Admin Console under Account & Billing > Enterprise ID.

`authentication.oauth2.client_id` (required)

The OAuth2 Client ID from the Box Custom App. Stored in AWS Secrets Manager.

`authentication.oauth2.client_secret` (required)

The OAuth2 Client Secret from the Box Custom App. Stored in AWS Secrets Manager.

> **Note**
>
> The `client_id` and `client_secret` values are retrieved from AWS Secrets Manager. These credentials can be found in the Box Developer Console under your Custom App's Configuration tab.

Broadcom Carbon Black integration configuration

Broadcom Carbon Black is a cloud-native endpoint protection platform (EPP) that combines next-generation antivirus (NGAV), behavioral detection, and endpoint detection and response (EDR). It provides continuous monitoring of endpoints and workloads from a single console to detect and stop ransomware, malware, and fileless attacks.

The Carbon Black Data Forwarder is a high-volume data streaming service designed to send alerts, endpoint events, watchlist hits, authentication events, auditlog events, and livequery results to Amazon S3.

Data Forwarder can be configured to forward events for one of the following types:

- alert 2.1.0
- auth.event 1.0.0
- endpoint.event 1.2.0
- watchlist.hit 1.0.0
- auditlog 1.0.0

- livequery 1.0.0

Topics

- Source configuration for Broadcom Carbon Black
- CloudWatch pipelines configuration for Broadcom Carbon Black

Source configuration for Broadcom Carbon Black

Integrating with Broadcom Carbon Black

CloudWatch Pipeline ingests Broadcom Carbon Black event data from Amazon S3. You configure Carbon Black Cloud to export event data to an Amazon S3 bucket using the Data Forwarder, then create a pipeline that reads from that bucket using Amazon SQS notifications.

To integrate CloudWatch Pipelines with Broadcom Carbon Black, complete the following high-level steps:

- Configure Carbon Black Cloud Data Forwarder to export events to an Amazon S3 bucket.
- Set up Amazon SQS notifications on the Amazon S3 bucket for new object creation.
- Create the required IAM role for pipeline access.
- Create a CloudWatch pipeline with Broadcom Carbon Black as the data source.
- Verify that data is flowing into the pipeline.

Instructions to set up Amazon S3 and Amazon SQS

Setting up the Carbon Black Data Forwarder to send logs to an Amazon S3 bucket involves several steps to create the required AWS resources (Amazon S3 bucket, Amazon SQS queue, IAM roles) and configure the Carbon Black Data Forwarder to send the desired log events.

1. Create an Amazon S3 bucket to store Carbon Black logs.

> **Important**
>
> You must select the same region as your Broadcom Carbon Black instance. See the Carbon Black documentation on the Broadcom developer website to determine which AWS region your Carbon Black product URL maps to.

> **Important**
>
> CloudWatch Pipelines will not be able to apply OCSF transformations or other processing to KMS encrypted logs. If processing is needed, do not set up KMS encryption.

For detailed instructions, see Configure the Destination: Use Amazon S3 in the Carbon Black Data Forwarder documentation on the Broadcom developer website.

2. Attach an IAM policy to the Amazon S3 bucket to grant the Forwarder permissions to write to your bucket (see *Example IAM policy for Amazon S3 bucket* below).

> **Note**
>
> The Principal ID ARN to use in the policy is based on the AWS region the Amazon S3 bucket is in. See the *Principal ID to allow by AWS Region* table below.

3. Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket. For more information, see Create an Amazon SQS queue.
4. Attach an IAM policy to the Amazon SQS queue to allow Amazon S3 to write event notifications to the Amazon SQS queue (see *Example IAM policy for Amazon SQS queue* below).
5. Configure the Amazon S3 bucket to generate "Object Create" event notifications to be sent to the Amazon SQS queue. For more information, see Enable event notifications on the S3 bucket with the SQS queue as a destination.
6. Create and configure the Carbon Black Data Forwarder, through the Carbon Black Console or API, by following the Carbon Black Create a Forwarder documentation. For complete documentation, see Carbon Black Data Forwarders on the Broadcom website.

Example IAM policy for Amazon S3 bucket

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AllowCarbonBlackForwarderWriteAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::132308400445:role/mcs-psc-prod-event-forwarder-
<aws-region>-event-forwarder"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::<bucket-name>[/<prefix>]/*"
        }
    ]
}
```

Principal ID to allow by AWS Region

AWS Region	Principal ID
US East (N. Virginia) us-east-1	`arn:aws:iam::132308400445:role/mcs-psc-prod-event-forwarder-us-east-1-event-forwarder`
Europe (Frankfurt) eu-central-1	`arn:aws:iam::132308400445:role/mcs-psc-prod-event-forwarder-eu-central-1-event-forwarder`
Asia Pacific (Tokyo) ap-northeast-1	`arn:aws:iam::132308400445:role/mcs-psc-prod-event-forwarder-ap-northeast-1-event-forwarder`
Asia Pacific (Sydney) ap-southeast-2	`arn:aws:iam::132308400445:role/mcs-psc-prod-event-forwarder-ap-southeast-2-event-forwarder`

Example IAM policy for Amazon SQS queue

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForS3SendMessage",
    "Statement": [
        {
            "Sid": "AllowS3SendMessage",
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "s3.amazonaws.com"
            },
            "Action": "sqs:SendMessage",
            "Resource": "arn:aws:sqs:<aws-region>:<aws-account-number>:<queue-name>",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:::<bucket-name>"
                },
                "StringEquals": {
                    "aws:SourceAccount": "<aws-account-number>"
                }
            }
        }
    ]
}
```

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Broadcom Carbon Black, choose Broadcom Carbon Black as the data source. After you create the pipeline, data will be available in the selected CloudWatch Logs log group. The log group will require a resource policy. The policy is automatically added if the pipeline is created through the console.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and the Broadcom Carbon Black log events (Endpoint Events v1.2.0, Alerts v2.1.0, Watchlist Hits v1.0.0, Auth Events v1.0.0, Audit Logs v1.0.0) that map to the following OCSF event classes.

When parseOCSF is enabled, if any log events aren't one of the supported log event types, the logs will be ingested in their raw form with a `@transformationError` field added and a `TransformationErrors` metric emitted.

File System Activity (1001)

Contains the following events:

- endpoint.event.filemod

Module Activity (1005)

Contains the following events:

- endpoint.event.moduleload
- endpoint.event.scriptload

Process Activity (1007)

Contains the following events:

- endpoint.event.procstart
- endpoint.event.procend
- endpoint.event.crossproc
- endpoint.event.apicall

Script Activity (1009)

Contains the following events:

- endpoint.event.fileless_scriptload

Detection Finding (2004)

Contains the following events:

- alert CB_ANALYTICS
- alert WATCHLIST
- alert INTRUSION_DETECTION_SYSTEM
- alert HOST_BASED_FIREWALL
- alert DEVICE_CONTROL
- alert CONTAINER_RUNTIME

Authentication (3002)

Contains the following events:

- auth.event.logonop

Network Activity (4001)

Contains the following events:

- endpoint.event.netconn
- endpoint.event.netconn_proxy

Registry Key Activity (win ext) (201001)

Contains the following events:

- endpoint.event.regmod (action ends in "_KEY" or action == "ACTION_SET_SECURITY")

Registry Value Activity (win ext) (201002)

Contains the following events:

- endpoint.event.regmod (action ends in "_VALUE")

CloudWatch pipelines configuration for Broadcom Carbon Black

The Broadcom Carbon Black setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Broadcom Carbon Black source with the following parameters:

```
source:
  s3:
    aws:
      region: "<aws_region>"
      sts_role_arn: "arn:aws:iam::<account_id>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "broadcom_carbonblack"
    default_bucket_owner: "<account_id>"
    bucket_owners:
      my-bucket: "<account_id>"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
```

```
      queue_url: "https://sqs.<aws_region>.amazonaws.com/<account_id>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be `sqs` to use Amazon SQS for Amazon S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: `broadcom_carbonblack`.

`aws.region` (required)

The AWS Region where the Amazon S3 bucket and Amazon SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing Amazon S3 and Amazon SQS resources.

`codec` (required)

Codec configuration for parsing Amazon S3 objects. Supports `csv`, `json`, `ndjson` codecs.

`compression` (optional)

Compression type of the Amazon S3 objects. Valid values are `none`, `gzip`, `automatic`. Defaults to `none`.

`sqs.queue_url` (required)

The complete Amazon SQS queue URL that receives Amazon S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either `retain_messages` or `delete_messages`. Default is `retain_messages`.

Check Point NGFW integration configuration

Check Point Next-Generation Firewall (NGFW), also known as Check Point Quantum, is an advanced on-premises firewall that combines traditional filtering with modern threat-prevention

technologies such as intrusion prevention, antivirus, and application control. CloudWatch pipelines allow ingestion of Check Point NGFW log data into CloudWatch Logs, providing scalable collection, processing, normalization, and integration with downstream AWS security and monitoring services.

Topics

- Source configuration for Check Point NGFW
- CloudWatch pipelines configuration for Check Point NGFW

Source configuration for Check Point NGFW

Integrating with Check Point NGFW

Check Point NGFW integration uses Amazon S3 and Amazon SQS to ingest log data into CloudWatch pipelines. Log data is forwarded from the Check Point appliance using the `cp_log_export` utility to a syslog receiver, then collected by Fluent Bit and delivered to an Amazon S3 bucket. Amazon SQS notifications alert the pipeline when new log objects arrive.

Prerequisites

- An AWS account with permissions to create Amazon S3 buckets, Amazon SQS queues, and IAM roles
- A Check Point NGFW appliance with access to the `cp_log_export` utility
- A Linux host with Fluent Bit installed (for log collection and delivery to Amazon S3)
- Network connectivity between the Check Point appliance, syslog receiver, and AWS

Log forwarding setup

Check Point NGFW exports logs using the `cp_log_export` utility, which forwards log data in syslog format to a designated receiver. Fluent Bit running on the syslog receiver host collects these logs and delivers them to Amazon S3 in NDJSON format.

Instructions to setup Amazon S3 and Amazon SQS

Complete the following steps to configure the Amazon S3 and Amazon SQS infrastructure for Check Point NGFW log ingestion.

Step 1: Create Amazon S3 bucket

Create an Amazon S3 bucket to store Check Point NGFW logs. The bucket must reside in the same AWS Region where you plan to create the CloudWatch pipeline.

Step 2: Create Amazon SQS queue

Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue receives notifications when new log files are added to the bucket.

Step 3: Connect Amazon S3 to Amazon SQS

Configure the Amazon S3 bucket to send event notifications for `s3:ObjectCreated:*` events to the Amazon SQS queue.

Step 4: Configure Amazon SQS queue policy

Configure the Amazon SQS queue policy to allow the Amazon S3 bucket to send messages to the queue. Apply the following policy to your Amazon SQS queue:

```
{
  "Version": "2012-10-17",
  "Id": "AllowS3ToSQS",
  "Statement": [
    {
      "Sid": "AllowS3BucketNotification",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "<account-id>"
        }
      }
    }
  ]
}
```

Step 5: Configure Check Point log export and Fluent Bit

On the Check Point appliance, configure `cp_log_export` to forward logs to your syslog receiver:

```
cp_log_export add name my_export target-server <syslog-host> target-port 514 protocol
 udp format syslog
```

On the syslog receiver host, install Fluent Bit and configure it to tail the Check Point NGFW log files and upload them to Amazon S3. Use the following example configuration:

```
########################################
# SERVICE SECTION
########################################
[SERVICE]
    Flush                     5
    Daemon                    Off
    Log_Level                 info
    storage.path              /etc/fluent-bit/db/storage
    storage.sync              normal
    storage.checksum          off
    storage.backlog.mem_limit 10M

########################################
# INPUT SECTION
########################################
[INPUT]
    Name                      tail
    Path                      /var/logs/<checkpoint_ip_address>/*.log
    Tag                       checkpoint.*
    Read_from_Head            True
    DB                        /etc/fluent-bit/db/checkpoint.db
    DB.Sync                   Normal
    storage.type              filesystem
    Rotate_Wait               30
    Refresh_Interval          60

########################################
# OUTPUT SECTION
########################################
[OUTPUT]
    Name                      s3
    Match                     checkpoint.*
    Bucket                    <<Bucket name>>
```

```
    Region                      <region>
    S3_Key_Format               /checkpoint-ngfw/%Y/%m/%d/%H/%M/%S.log.gz
    compression                 gzip
    use_put_object              On
    store_dir                   /etc/fluent-bit/db/s3
    total_file_size             1M
    log_key                     log
    upload_timeout              60
```

Note

- Logs from `cp_log_export` arrive in JSON format through syslog to the receiver host.
- `log_key log` sends only the value of the `log` key to Amazon S3.
- `storage.path` and `store_dir` provide local buffering.
- The DB parameter tracks file read positions to prevent duplicate ingestion.
- Replace `<checkpoint_ip_address>` with the actual IP address.
- Replace `<<Bucket name>>` with your actual Amazon S3 bucket name.

Step 6: IAM permissions

Create an IAM policy with the following permissions for the Fluent Bit host to write objects to the Amazon S3 bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FluentBitS3Write",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
```

```
        "arn:aws:s3:::amzn-s3-demo-bucket/checkpoint-ngfw/*"
      ]
    }
  ]
}
```

Step 7: Verify

Verify the setup by confirming that log files appear in the Amazon S3 bucket and that Amazon SQS notifications are being generated. Check the Amazon SQS queue for messages indicating new object creation events.

Configuring the CloudWatch pipeline

- Choose Check Point NGFW as the data source when creating the pipeline.
- Provide the Amazon SQS queue URL and IAM role ARN.
- Select the destination CloudWatch Logs log group.
- After you create the pipeline, data will be available in the selected log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. The following table shows the mapping between Check Point NGFW event types and OCSF event classes.

Event name	OCSF event class
Firewall	Network Activity [4001]
URL Filtering	Network Activity [4001]
HTTPS Inspection	Network Activity [4001]
Application Control	Network Activity [4001]
Identity Awareness	Network Activity [4001]
Anti Virus	Detection Finding [2004]
Anti Spam	Detection Finding [2004]

Event name	OCSF event class
IPS	Detection Finding [2004]
Anti Bot	Detection Finding [2004]
Threat Emulation	Detection Finding [2004]
Audit	Entity Management [3004]
DLP	Data Security Finding [2006]

CloudWatch pipelines configuration for Check Point NGFW

The Check Point NGFW setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "checkpoint_nextgenerationfirewall"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. Must be "checkpoint_nextgenerationfirewall".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

Cisco Duo integration configuration

Cisco Duo is a cloud-based identity and access management (IAM) platform that provides secure access through multi-factor authentication (MFA), device trust, and adaptive authentication policies. CloudWatch Pipelines use the Duo Admin API to retrieve audit logs, authentication activity, administrator actions, and policy-related events from your Cisco Duo environment.

Topics

- Source configuration for Cisco Duo
- CloudWatch pipelines configuration for Cisco Duo

Source configuration for Cisco Duo

Integrating with Cisco Duo

CloudWatch Pipeline uses the Cisco Duo Admin API to retrieve audit logs, authentication activity, administrator actions, and policy-related events from your Cisco Duo environment. The Admin API

exposes REST endpoints that allow fetching time-filtered event data for security monitoring and analysis.

To integrate CloudWatch Pipelines with Cisco Duo, complete the following high-level steps:

- Create an Admin API integration in the Duo Admin Panel.
- Note the Integration Key, Secret Key, and API Hostname.
- Store the credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with Cisco Duo as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- An active Cisco Duo account with Admin API access
- A Cisco Duo auth credential with Admin API access
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager

Authenticating with Cisco Duo

Cisco Duo uses HMAC Authentication with an integration key and secret key, combined with request signing. The pipeline uses these credentials to authenticate each API request to the Duo Admin API.

Configure Authentication for Cisco Duo

To configure authentication credentials for the pipeline:

1. Log in to the Duo Admin Panel and navigate to Applications > Protect an Application.
2. Search for "Admin API" and create a new integration.
3. Note the Integration Key, Secret Key, and API Hostname provided by Duo.
4. Grant the "Grant read log" permission to the integration.
5. Store the `integration_key` and `secret_key` in AWS Secrets Manager.

Configuring the CloudWatch Pipeline

To configure the pipeline, choose Cisco Duo as the data source. Provide the `api_host`, `integration_key`, and `secret_key`. After you create and activate the pipeline, log data from Cisco Duo will begin flowing into the selected CloudWatch Logs log group.

The following optional parameters are available:

- **Page sizes** – Default is 100 for all streams. Valid ranges vary by stream.
- **Polling intervals** – Default is `P1D` (one day). Valid range is `PT30M` to `P15D`.
- **Backfill durations** – Default is `P180D` (180 days). Valid range is `PT1H` to `P180D`.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports the following OCSF event classes:

- **Base Event [0]** – Telephony Logs
- **Authentication [3002]** – Authentication Logs
- **Entity Management [3004]** – Activity Logs
- **Device Inventory Info [5001]** – Registered Devices, Phones
- **User Inventory Info [5003]** – Users

CloudWatch pipelines configuration for Cisco Duo

Collects log data from Cisco Duo using HMAC authentication through the Duo Admin API.

Configure the Cisco Duo source with the following parameters:

```
source:
  cisco_duo:
    api_host: "api-XXXXXXXX.duosecurity.com"
    authentication:
      hmac:
        integration_key: "${{aws_secrets:<secret-name>:integration_key}}"
        secret_key: "${{aws_secrets:<secret-name>:secret_key}}"
    range: "P30D"
```

Parameters

`api_host` (required)

The Duo API hostname for your organization (for example, `api-XXXXXXXX.duosecurity.com`).

`authentication.hmac.integration_key` (required)

The Duo Admin API integration key, stored in AWS Secrets Manager.

`authentication.hmac.secret_key` (required)

The Duo Admin API secret key, stored in AWS Secrets Manager.

`range` (optional)

The historical time period for backfilling data. Uses ISO 8601 duration format. Minimum is `PT1H`, maximum is `P180D`. Default is `P180D`.

Cisco FTD integration configuration

Cisco Firepower Threat Defense (FTD) is a next-generation firewall platform that provides advanced threat protection, intrusion prevention, and deep visibility into network traffic. It generates security logs including connection events, intrusion detections, file and malware activity, and system events, which can be exported for centralized monitoring and analysis.

Topics

- Source configuration for Cisco FTD
- CloudWatch pipelines configuration for Cisco FTD

Source configuration for Cisco FTD

Integrating with Cisco FTD

Cisco FTD exports logs using the syslog protocol. These logs are collected by a syslog server, processed using Fluent Bit, and delivered to a customer-managed Amazon S3 bucket. CloudWatch pipelines read from this bucket using Amazon SQS notifications for new object events.

To integrate Cisco FTD with CloudWatch Logs, you must configure both the source and the pipeline. First, set up the syslog server, Fluent Bit, Amazon S3, and Amazon SQS to receive and

store data from Cisco FTD. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Prerequisites

Before you begin, make sure you have the following:

- Access to Cisco FTD with permissions to configure syslog export (through Firepower Management Center or CLI)
- A Linux-based syslog server (Amazon Linux 2 or Amazon Linux 2023 recommended) with network connectivity to your Cisco FTD appliance
- An AWS account with permissions to create and manage CloudWatch pipelines
- An AWS account with permissions to create and manage Amazon S3 buckets, Amazon SQS queues, and IAM roles

Instructions to set up Syslog Server, Fluent Bit, Amazon S3 and Amazon SQS

1. **Create an Amazon S3 Bucket**

   Create an Amazon S3 bucket in the same AWS region where you plan to run the pipeline. Enable server-side encryption and block public access. Reference: Creating a general purpose bucket

2. **Create an IAM User with Programmatic Access for Fluent Bit**

   In IAM, create a user with programmatic (CLI) access for Fluent Bit to write logs to Amazon S3. Attach a policy scoped to the target bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket-name>",
                "arn:aws:s3:::<bucket-name>/*"
            ]
```

```
        }
    ]
}
```

3. **Create an Amazon SQS Queue**

   Create a standard Amazon SQS queue in the same AWS region as the Amazon S3 bucket. Note the Queue URL and Queue ARN. Reference: Creating an Amazon SQS queue

4. **Configure Amazon SQS Queue Policy**

   Add a resource policy to the Amazon SQS queue allowing Amazon S3 to send event notifications:

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {"Service": "s3.amazonaws.com"},
        "Action": "sqs:SendMessage",
        "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>",
        "Condition": {
            "ArnEquals": {"aws:SourceArn": "arn:aws:s3:::<bucket-name>"},
            "StringEquals": {"aws:SourceAccount": "<account-id>"}
        }
    }]
}
```

5. **Enable Amazon S3 Event Notifications**

   Navigate to the Amazon S3 bucket > Properties > Event notifications. Create a notification for `s3:ObjectCreated:*` events. Select the Amazon SQS queue created in Step 3 as the destination. Reference: Enabling event notifications

6. **Configure Cisco FTD, the Syslog Server, and Fluent Bit**

   Configure Cisco FTD to forward logs over syslog (UDP 514/5514) to the syslog server, set up the syslog server (for example, rsyslog on Amazon Linux) to receive them, and install Fluent Bit to compress (gzip) and upload the logs to the Amazon S3 bucket created in Step 1.

   Use the following Fluent Bit configuration as a reference:

```
[SERVICE]
```

```
    Flush             1
    Daemon            Off
    Log_Level         info
    Parsers_File      /etc/fluent-bit/parsers.conf
    storage.path      /tmp/fluent-bit/storage

[INPUT]
    Name              tail
    Path              /var/log/cisco-ftd.log,/var/log/messages
    Tag               cisco.ftd
    Read_from_Head    True
    Refresh_Interval  5
    Mem_Buf_Limit     50MB
    Skip_Long_Lines   On
    DB                /tmp/fluent-bit/flb.db

[OUTPUT]
    Name              s3
    Match             cisco.ftd
    bucket            <bucket-name>
    region            <region>
    total_file_size   5M
    upload_timeout    1m
    store_dir         /tmp/fluent-bit/s3
    s3_key_format     /cisco-ftd/%Y/%m/%d/%H/%M/%S-$UUID.gz
    compression       gzip
    log_key           log
```

Replace <bucket-name> and <region> with the values from Step 1. Ensure Fluent Bit has AWS credentials (the IAM user from Step 2, or an instance role) with permission to write to the bucket.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Cisco FTD, choose Cisco FTD as the data source. Provide the Amazon SQS queue URL and IAM role ARN. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following events that map to Authentication (3002), Tunnel Activity (4014), Network Activity (4001), Detection Finding

(2004), and Authorize Session (3003). Events that are not listed but pulled are not mapped to OCSF and will be forwarded to the sink as raw logs.

Authentication contains the following Syslog IDs:

109011, 109012, 109013, 109026, 109027, 109039, 109201, 109202, 109203, 109204, 109205, 109206, 109207, 109208, 109209, 109210, 109211, 109212, 109213, 113001, 113004, 113005, 113006, 113007, 113008, 113009, 113010, 113012, 113013, 113014, 113016, 113017, 113019, 113020, 113021, 113022, 113023, 113024, 113025, 113026, 113027, 113028, 113029, 113033, 113035, 113037, 113038, 113039, 113040, 214001, 308001, 320001, 324302, 324303, 333001, 333002, 333003, 333004, 333005, 333006, 333007, 333008, 333009, 333010, 334001, 334002, 334003, 334004, 334005, 334006, 334007, 334008, 334009, 611101, 611102, 611103, 611104, 611316, 611317, 611318, 611319, 716036, 716038, 716039, 716040, 716160, 717001, 717009, 717010, 717012, 717013, 717014, 717015, 717016, 717017, 717018, 717021, 717022, 717024, 717025, 717026, 717027, 717028, 717029, 717030, 717031, 717032, 717033, 717034, 717035, 717036, 717037, 717038, 717053, 772002, 772003, 772004, 772005, 772006

Tunnel Activity contains the following Syslog IDs:

316001, 316002, 602303, 602304, 602305, 602306, 603110, 611309, 611315, 702307, 713041, 713049, 713050, 713056, 713060, 713061, 713073, 713074, 713075, 713076, 713092, 713099, 713113, 713120, 713123, 713136, 713169, 713187, 713194, 713195, 713196, 713206, 713207, 713218, 713227, 713239, 713259, 713262, 713276, 715009, 715013, 715080, 716001, 716002, 716006, 716007, 716009, 716023, 716057, 716058, 716059, 716060, 716166, 718048, 718049, 718050, 718051, 722020, 722029, 722030, 722031, 722032, 722033, 722034, 722037, 722038, 722041, 722045, 722046, 722047, 722048, 722049, 722050, 722054, 724001, 724002, 750001, 750002, 750003, 750004, 750005, 750006, 750007, 750008, 750009, 750010, 750011, 750012, 750013, 750014, 751001, 751002, 751003, 751004, 751005, 751006, 751007, 751008, 751009, 751010, 751011, 751012, 751013, 751014, 751015, 751016, 751017, 751018, 751019, 751020, 751021, 751022, 751023, 751024, 751025, 751026, 751027, 751031, 752002, 752012, 752015, 752016, 752017, 840001

Network Activity contains the following Syslog IDs:

430002, 430003, 106029, 302003, 302004, 302012, 302013, 302015, 302017, 302020, 302022, 302024, 302026, 302033, 302037, 302303, 305009, 305011, 609001, 302014, 302016, 302018, 302021, 302023, 302025, 302027, 302038, 302304, 305010, 305012, 419006, 507001, 609002, 500005, 110002, 110003, 201005, 201006, 302034, 302311, 305006, 305016, 305021, 305023, 106001, 106002, 106006, 106007, 106010, 106011, 106012, 106013, 106014, 106015, 106016,

106017, 106018, 106020, 106021, 106022, 106023, 106027, 106103, 201002, 201003, 201004, 201008, 201009, 201010, 201011, 201012, 201013, 209003, 209004, 209005, 209006, 302302, 313001, 313004, 313005, 313008, 313009, 322001, 407001, 407002, 407003, 418001, 419001, 421001, 424001, 424002, 446003, 448001, 500003, 500004, 509001, 710003, 710004, 710005, 710006, 767001, 815002, 106100, 106102, 302010, 302310, 305013, 419002, 419003, 419004, 419005, 421007, 507003, 775002, 805001, 805002, 805003, 815004, 852001, 852002

Detection Finding contains the following Syslog IDs:

430001, 430004, 430005

Authorize Session contains the following Syslog IDs:

109100, 109101, 109102, 109103, 109104, 113003, 113011, 113034, 113036, 501101, 634001, 734001, 734003, 113015, 113018, 113030, 113031, 113032, 113041, 113042, 730002, 734002, 109016, 109018, 109019, 109020, 109029, 109030, 109032, 109033, 109034, 109035, 109036, 109037, 109038, 610101, 734004

CloudWatch pipelines configuration for Cisco FTD

The Cisco FTD setup reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Cisco FTD source with the following parameters:

```
source:
  s3:
    aws:
      region: "<region>"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      newline:
    data_source_name: "cisco_ftd"
    default_bucket_owner: "<account-id>"
    bucket_owners:
      <bucket-name>: "<account-id>"
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.<region>.amazonaws.com/<account>/<queue-name>"
```

Parameters

`sqs.queue_url` (required)

Amazon SQS queue receiving Amazon S3 event notifications.

`data_source_name` (required)

Set to `cisco_ftd`.

`aws.region` (required)

Region of Amazon S3 and Amazon SQS.

`aws.sts_role_arn` (required)

IAM role to assume for Amazon S3/Amazon SQS access.

`notification_type` (required)

Set to `sqs`.

`codec` (required)

Codec for parsing Amazon S3 objects. Cisco FTD uses `newline`.

`bucket_owners` (optional)

Mapping of Amazon S3 bucket to AWS account ID.

`default_bucket_owner` (optional)

Default AWS account ID.

`compression` (optional)

Default none.

Note

The `sts_role_arn` role must have permissions to read from Amazon S3 and receive/delete Amazon SQS messages. See the pipeline IAM reference for the required trust policy and permissions policy.

Cisco ISE integration configuration

Cisco Identity Services Engine (ISE) is the Cisco enterprise Network Access Control (NAC) and identity policy engine, providing AAA services (Authentication, Authorization, Accounting) using RADIUS and TACACS+ across wired, wireless, and VPN connections. ISE enforces who and what can connect to the network, under what conditions, and with what privileges, while providing rich session context including user identity, device type, posture, and location to other security tools.

Use this connector to collect Passed Authentications, Failed Attempts, Authentication Flow Diagnostics, Identity Store Diagnostics, Policy Diagnostics, RADIUS Accounting, Guest, and Administrative and Operational Audit events. Use CloudWatch pipelines to ingest Cisco ISE log data into CloudWatch Logs for scalable collection, processing, normalization, and integration with downstream AWS security and monitoring services.

Topics

- Source configuration for Cisco ISE
- CloudWatch pipelines configuration for Cisco ISE

Source configuration for Cisco ISE

Integrating with Cisco ISE

CloudWatch Pipeline ingests Cisco ISE log data from Amazon S3. You configure ISE to forward syslog to a Fluent Bit host, which compresses and uploads the logs to Amazon S3. The pipeline then reads from Amazon S3 using Amazon SQS notifications.

To integrate CloudWatch Pipelines with Cisco ISE, complete the following high-level steps:

- Configure ISE Remote Logging Target to forward syslog to a Fluent Bit host.
- Assign the Remote Target to the desired Logging Categories.
- Install and configure Fluent Bit to receive syslog and upload to Amazon S3.
- Create an Amazon S3 bucket and Amazon SQS queue with event notifications.
- Create a CloudWatch pipeline with Cisco ISE as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- Active Cisco ISE deployment (version 2.x or 3.x) with administrative access to configure Remote Logging Targets and Logging Categories
- Network connectivity from ISE nodes to the Fluent Bit host (TCP port configurable, default 1514)
- Fluent Bit installed on a host reachable from all ISE nodes generating logs
- AWS credentials configured for Fluent Bit to write to Amazon S3
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create and manage CloudWatch Logs log groups

Instructions to set up Cisco ISE, Fluent Bit, Amazon S3 and Amazon SQS

1. **Configure ISE Remote Logging Target**

   In the ISE Admin Console, navigate to **Administration > System > Logging > Remote Logging Targets > Add**. Configure the following settings:

   - **Name:** Descriptive name (for example, `FluentBit_S3_Collector`)
   - **Target Type:** `TCP Syslog` or `Secure Syslog` (TCP+TLS) recommended for reliability
   - **Status:** Enabled
   - **Host/IP Address:** Fluent Bit host IP or FQDN
   - **Port:** Must match Fluent Bit listener port (for example, 1514)
   - **Maximum Length:** Set to **8192** (default 1024 will truncate messages)
   - **Comply to RFC 3164:** Check this box
2. **Assign Remote Target to Logging Categories**

   Navigate to **Administration > System > Logging > Logging Categories**. For each desired category, move the remote target from "Available" to "Selected".

   - **Categories from PSN nodes only:** AAA Audit, AAA Diagnostics, Accounting, Profiler, Posture, External MDM, Passive ID
   - **Categories from all nodes:** Administrative and Operational Audit, System Diagnostics, System Statistics

   Ensure Fluent Bit is reachable from all ISE nodes that generate the selected categories.
3. **Install and configure Fluent Bit**

   Install Fluent Bit on the designated log collection host. Configure Fluent Bit with a TCP input to receive syslog from ISE and an Amazon S3 output plugin:

```
[SERVICE]
    Flush        5
    Log_Level    info

[INPUT]
    Name         tcp
    Tag          cisco.ise
    Listen       0.0.0.0
    Port         1514
    Format       none
    Separator    \n

[OUTPUT]
    Name         s3
    Match        cisco.ise
    bucket       <your-bucket-name>
    region       <your-region>
    s3_key_format /cisco-ise-logs/$TAG/%Y/%m/%d/%H/%M/$UUID.gz
    total_file_size 5M
    upload_timeout 60s
    use_put_object On
    compression  gzip
    content_type application/gzip
    store_dir    /tmp/fluent-bit-s3
    log_key      log
```

> **Note**
>
> Maximum message length in ISE MUST be set to 8192 to avoid truncation of verbose ISE syslog messages. Fluent Bit uses TCP input (not the built-in syslog input) because ISE's syslog format is not fully RFC 3164 compliant. The `log_key` parameter ensures only the raw log line is written to Amazon S3 (without Fluent Bit metadata).

4. **Create Amazon S3 bucket and Amazon SQS queue**

   Create an Amazon S3 bucket in your desired AWS region to receive Cisco ISE logs. Create an Amazon SQS queue in the same region and configure the Amazon S3 bucket to send "Object Create" event notifications to the Amazon SQS queue.

5. **Verify log delivery**

Use ISE TCP Dump (**Operations > Troubleshoot > TCP Dump**) filtered by Fluent Bit IP to confirm log delivery from ISE. Verify objects appear in the Amazon S3 bucket under the configured prefix.

Configuring the CloudWatch Pipeline

When configuring the pipeline, choose **Cisco ISE** as the data source. Configure the Amazon S3 bucket and Amazon SQS queue details and select the destination log group. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Data flow

```
Cisco ISE -> Fluent Bit (TCP Syslog) -> Amazon S3 -> Amazon SQS Notification ->
 CloudWatch Pipeline -> CloudWatch Logs
```

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports **OCSF schema version v1.5.0**. Cisco ISE syslog events are parsed using a custom ISE syslog parser and mapped to OCSF classes based on the ISE logging category and message code.

ISE Event Type	OCSF Event Class	Description
Failed Attempts	Authentication (3002)	Failed RADIUS authentication and authorization attempts
Passed Authentication	Authentication (3002)	Successful RADIUS authentication and authorization events
Radius Accounting	Network Activity (4001), Tunnel Activity (4014)	RADIUS accounting start, stop, and interim update events (codes 3000-3004). RADIUS tunnel-related accounting events (codes 3005-3010).
Policy Diagnostics	Authentication (3002)	Policy evaluation and matching diagnostic events

ISE Event Type	OCSF Event Class	Description
Identity Store Diagnostics	Authentication (3002)	Identity store lookup and validation diagnostic events
Authentication Flow Diagnostics	Authentication (3002)	Authentication flow processing diagnostic events
Administrative and Operational Audit	Authentication (3002), Account Change (3001), Entity Management (3004), Application Lifecycle (6002), Application Error (6008), API Activity (6003), Network Activity (4001), Detection Finding (2004)	Administrative actions, system operations, and configuration changes mapped by message code
Guest	Web Resources Activity (6004)	Guest portal access and lifecycle events (codes 86001, 86005, 86012, 86021-86023, 86030-86031)

The following event types are ingested but not currently mapped to OCSF classes (forwarded as raw logs):

- Radius Diagnostics
- Administrator Authentication and Authorization
- Posture and Client Provisioning Audit
- Posture and Client Provisioning Diagnostics
- Profiler
- System Diagnostics
- Distributed Management
- Internal Operations Diagnostics
- System Statistics

Events that do not match any OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

CloudWatch pipelines configuration for Cisco ISE

The Cisco ISE setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Cisco ISE source with the following parameters:

```
source:
  s3:
    aws:
      region: "<region>"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      multiline:
        continuation_line_end_pattern: "\\\\$"
        omit_matched_section: true
        line_separator: ""
    data_source_name: "cisco_ise"
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.<region>.amazonaws.com/<account>/<queue-name>"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be `sqs` to use Amazon SQS for Amazon S3 event notifications.

`data_source_name` (required)

Identifies the data source. Value: `cisco_ise`.

`aws.region` (required)

The AWS Region where the Amazon S3 bucket and Amazon SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing Amazon S3 and Amazon SQS resources.

`codec` (required)

Codec configuration for parsing Amazon S3 objects. Must be `multiline` for ISE syslog format.

`compression` (optional)

Compression type of Amazon S3 objects. Valid values: `gzip`, none. Default: none.

`sqs.queue_url` (required)

The complete Amazon SQS queue URL that receives Amazon S3 bucket notifications.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Valid values: `retain_messages`, `delete_messages`. Default: `retain_messages`.

Cisco Meraki integration configuration

Cisco Meraki is a cloud-managed IT platform by Cisco that provides centralized management of enterprise networking infrastructure — including security appliances (MX), wireless access points (MR), switches (MS), cameras (MV), sensors (MT), and mobile device management (SM) — through a single dashboard. CloudWatch pipelines uses the Meraki Dashboard REST API v1 to retrieve security events, configuration changes, assurance alerts, and network events from your Meraki organization.

Topics

- Source configuration for Cisco Meraki
- CloudWatch pipelines configuration for Cisco Meraki

Source configuration for Cisco Meraki

Integrating with Cisco Meraki

To integrate Cisco Meraki with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your Cisco Meraki source by configuring API access to the Meraki Dashboard API to retrieve data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Authenticating with the Meraki Dashboard API

To retrieve events from Cisco Meraki, CloudWatch pipelines needs to authenticate with your Meraki organization. Cisco Meraki supports API key access.

API key

- Generate an API key from the Meraki Dashboard. Navigate to your profile and select **API access** to generate a new API key.
- In AWS Secrets Manager, create a secret and store the API key under the key `api_key`.
- API keys are permanent and can be scoped down if needed. Make sure the API key has at least read-only access to the specific organization and the APIs.

For more information on Meraki API authorization, see Meraki API Authorization.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Cisco Meraki, choose Cisco Meraki as the data source. Fill in the required information including the Organization ID and the secret where your credentials are stored. You can find your organization ID in the Meraki Dashboard under **Organization > Settings**, or by calling `GET /organizations` through the Cisco Meraki API. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Cisco Meraki Dashboard API events that map to Network Activity (4001), API Activity (6003), and Detection Finding (2004).

Network Activity (4001)

Network Activity maps to Meraki Security Events — IDS/IPS alerts (Snort-based), malware detections through Advanced Malware Protection (AMP), and file scan results from MX security appliances.

API endpoint: `GET /organizations/{organizationId}/appliance/security/events`

API Activity (6003)

API Activity maps to Meraki Configuration Changes — an audit log of all administrative actions capturing who changed what configuration, when, and the before/after values. Covers changes made through the Dashboard UI and API.

API endpoint: `GET /organizations/{organizationId}/configurationChanges`

Detection Finding (2004)

Detection Finding maps to Meraki Assurance Alerts — alerts produced by Meraki's monitoring engine that analyze device telemetry and produce discrete alerts with IDs, severity, and a lifecycle status.

API endpoint: `GET /organizations/{organizationId}/assurance/alerts`

CloudWatch pipelines configuration for Cisco Meraki

Collects log data from the Cisco Meraki Dashboard REST API v1 using API key authentication.

Configure the Cisco Meraki source with the following parameters:

```
source:
  cisco_meraki:
    organization_id: "<meraki-organization-ID>"
    range: "P7D"
    authentication:
      api_key: "${{aws_secrets:<secret-name>:api_key}}"
```

Parameters

`organization_id` (required)

The Cisco Meraki organization ID. Each pipeline collects data from a single organization. You can find your organization ID in the Meraki Dashboard under **Organization > Settings**, or by calling `GET /organizations`.

`authentication.api_key` (required)

Cisco Meraki API key, stored as a key/value pair. From the preceding example, the name of the key will be `api_key`.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

> **Note**
>
> Store sensitive credentials like API keys in AWS Secrets Manager and reference them using the `${{aws_secrets:secret-name:key}}` syntax in your configuration.

Cisco Umbrella

To integrate Cisco Umbrella Data Replicator with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your Cisco Umbrella source by configuring Amazon S3 and Amazon SQS to receive data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Topics

- Source configuration for Cisco Umbrella
- Pipeline configuration for Cisco Umbrella

Source configuration for Cisco Umbrella

Integrating with Cisco Umbrella

Cisco Umbrella is a cloud-delivered security platform that provides secure internet access and threat protection across all devices, locations, and users. It uses DNS-layer security, web filtering, and cloud-delivered firewall features to block malicious domains and prevent cyberattacks before they reach your network. CloudWatch pipelines enables you to collect this data in CloudWatch Logs.

Instructions to setup Amazon S3 and Amazon SQS

Configuring Cisco Umbrella to send logs to an Amazon S3 bucket involves several steps, primarily focused on setting up the Amazon S3 bucket, Amazon SQS queue, and IAM roles, then configuring the CloudWatch pipeline.

- Make sure Cisco Umbrella logs environment exporter is configured with S3. This is typically found under Admin → Logs Management in the Cisco Umbrella console.
- Amazon S3 bucket that stores the Cisco Umbrella logs should reside in the same AWS region as your CloudWatch pipeline.

- Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to the Amazon SQS queue you created in the previous step.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Cisco Umbrella, choose Cisco Umbrella as the data source. After filling in the required information and creating the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Cisco Umbrella events that map to DNS Activity (4003), Network Activity (4001), Data Security Finding (2006), and Entity Management (3004). Each event comes from a source as mentioned below.

DNS Activity contains the following actions:

- DNS Logs

Network Activity contains the following actions:

- Cloud Firewall Logs
- Secure Gateway logs
- IPS logs

Data Security Finding contains the following actions:

- DLP (Data Loss Prevention) logs

Entity Management contains the following actions:

- Admin Audit logs

Pipeline configuration for Cisco Umbrella

The Cisco Umbrella setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Cisco Umbrella source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      csv:
    data_source_name: "cisco_umbrella"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "cisco_umbrella".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

CrowdStrike integration configuration

To integrate CrowdStrike Falcon Data Replicator with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your CrowdStrike source by configuring Amazon S3 and Amazon SQS to receive FDR data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Topics

- Source configuration for CrowdStrike
- CloudWatch pipelines configuration for CrowdStrike

Source configuration for CrowdStrike

Integrating with CrowdStrike Falcon

CrowdStrike Falcon Data Replicator (FDR) delivers and enriches endpoint, cloud workload and identity data with the CrowdStrike Security Cloud and world-class artificial intelligence (AI), enabling your team to derive actionable insights to improve security operations center (SOC) performance. Amazon CloudWatch Logs enables you to collect this data in CloudWatch Logs.

Instructions to setup Amazon S3 and Amazon SQS

Configuring CrowdStrike FDR to send logs to an Amazon S3 bucket involves several steps, primarily focused on setting up the Amazon S3 bucket, Amazon SQS queue, IAM roles, and then configuring the Amazon Telemetry Pipeline.

- Make sure CrowdStrike FDR is enabled within your CrowdStrike Falcon environment. This typically requires a specific license and may involve working with CrowdStrike support.
- Amazon S3 bucket that stores the CrowdStrike logs should reside in the same AWS region where the FDR is enabled.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from CrowdStrike FDR, choose CrowdStrike as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and the CrowdStrike FDR actions that maps to Detection Findings (2004) and Process Activity (1007).

Detection Findings

Detection Findings contains the following actions:

- CloudAssociateTreeIdWithRoot
- CustomIOADomainNameDetectionInfoEvent
- TemplateDetectAnalysis

Process Activity

Process Activity contains the following actions:

- ActiveDirectoryIncomingPsExecExecution2
- AndroidIntentSentIPC
- AssociateTreeIdWithRoot
- AutoRunProcessInfo
- BamRegAppRunTime
- BlockThreadFailed
- BrowserInjectedThread
- CidMigrationConfirmation
- CodeSigningAltered
- CommandHistory
- CreateProcessArgs
- CreateThreadNoStartImage
- CriticalEnvironmentVariableChanged
- CsUmProcessCrashAuxiliaryEvent
- CsUmProcessCrashSummaryEvent
- CustomIOABasicProcessDetectionInfoEvent
- DebuggableFlagTurnedOn
- DebuggedState
- DllInjection
- DocumentProgramInjectedThread
- EarlyExploitPivotDetect
- EndOfProcess
- EnvironmentVariablesChanged
- FalconProcessHandleOpDetectInfo
- FlashThreadCreateProcess
- IdpWatchdogRemediationActionTaken
- InjectedThread
- InjectedThreadFromUnsignedModule

- IPCDetectInfo
- JavaInjectedThread
- KillProcessError
- LsassHandleFromUnsignedModule
- MacKnowledgeActivityEnd
- MacKnowledgeActivityStart
- NamespaceChanged
- PcaAppLaunchEntry
- PcaGeneralDbEntry
- PrivilegedProcessHandle
- PrivilegedProcessHandleFromUnsignedModule
- ProcessActivitySummary
- ProcessBlocked
- ProcessControl
- ProcessDataUsage
- ProcessExecOnPackedExecutable
- ProcessHandleOpDetectInfo
- ProcessHandleOpDowngraded
- ProcessInjection
- ProcessPatternTelemetry
- ProcessRollup
- ProcessRollup2
- ProcessRollup2Stats
- ProcessSelfDeleted
- ProcessSessionCreated
- ProcessSubstituteUser
- ProcessTokenStolen
- ProcessTrace
- ProcessTreeCompositionPatternTelemetry

- PtTelemetry
- PtyCreated
- QueueApcEtw
- ReflectiveDllOpenProcess
- RegisterRawInputDevicesEtw
- RemediationActionKillProcess
- RemediationMonitorKillProcess
- RuntimeEnvironmentVariable
- ScriptControlDotNetMetadata
- ScriptControlErrorEvent
- ServiceStarted
- SessionPatternTelemetry
- SetThreadCtxEtw
- SetWindowsHook
- SetWindowsHookExEtw
- SetWinEventHookEtw
- ShellCommandLineInfo
- SruApplicationTimelineProvider
- SudoCommandAttempt
- SuspectCreateThreadStack
- SuspendProcessError
- SuspiciousPrivilegedProcessHandle
- SuspiciousUserFontLoad
- SuspiciousUserRemoteAPCAttempt
- SyntheticPR2Stats
- SyntheticProcessRollup2
- SyntheticProcessTrace
- SystemTokenStolen
- TerminateProcess

- ThreadBlocked
- UACAxisElevation
- UACCOMElevation
- UACExeElevation
- UACMSIElevation
- UmppcBypassSuspected
- UnexpectedEnvironmentVariable
- UserAssistAppLaunchInfo
- UserSetProcessBreakOnTermination
- WmiCreateProcess
- WmiFilterConsumerBindingEtw

CloudWatch pipelines configuration for CrowdStrike

The CrowdStrike setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "crowdstrike_falcon"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "crowdstrike_falcon".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

Drupal Core integration configuration

Drupal Core is the foundational open-source web application framework built on PHP that provides the base platform for building websites, applications, and digital experiences.

Topics

- Source configuration for Drupal Core

- CloudWatch pipelines configuration for Drupal Core

Source configuration for Drupal Core

Integrating with Drupal Core

Drupal Core is the foundational open-source web application framework built on PHP that provides the base platform for building websites, applications, and digital experiences. CloudWatch Pipeline uses the custom View-based REST API to retrieve audit log data — including content changes, user authentication events, and administrative actions — from your Drupal Core site. The API enables access to time-filtered log data through REST endpoints, allowing retrieval of activity records scoped to a configurable time window.

Authenticating with Drupal Core

To read the logs, the pipeline needs to authenticate with your Drupal Core site. The plugin supports Basic Authentication (HTTP Basic Auth using a username and password).

Configure Basic Authentication for Drupal Core

- Log in to your Drupal Core admin interface and navigate to Administration → Extend (`/admin/modules`).
- Enable the following modules: RESTful Web Services, Serialization, HTTP Basic Authentication, and Views. Choose Install.
- Install and enable the Admin Audit Trail module by using Composer (`composer require drupal/admin_audit_trail`) and run `drush en admin_audit_trail -y && drush cr` to activate it.
- Navigate to Structure → Views and create a new View named `Audit Logs API`. Set Show to `Log entries`, enable Provide a REST export, and set the REST export path to `/api/v1/audit-logs`.
- In the View editor, add two exposed Watchdog: Timestamp filters — one with operator `is greater than or equal to` and filter identifier `starttime`, and another with operator `is less than` and filter identifier `endtime`.
- In the REST EXPORT SETTINGS section of the View, choose Authentication and enable `basic_auth`.
- Navigate to People → Permissions and grant the roles that need API access the Access admin audit trail and Administer REST resource configuration permissions. Save the View.

- In AWS Secrets Manager, create a secret and store the Drupal Core username under the key `username` and the account password under the key `password`.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose Drupal Core as the data source. Fill in the required information:

- **Domain** — The base URL of your Drupal Core site (for example, `https://your-drupal-site.example.com`).
- **API Endpoint** — The path to the View REST export endpoint (for example, `/api/v1/audit-logs`). Must start with `/`.
- **Range** — Specify the lookback duration in ISO 8601 format (for example, `PT21H` for the last 21 hours, `P7D` for the last 7 days). The default is 0 hours, and the maximum is 90 days.

After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms events that map to Authentication (3002), Entity Management (3004), HTTP Activity (4002), and Application Lifecycle (6002). Events that are not listed are not mapped to OCSF and will be forwarded to the sink as raw logs.

Authentication contains the following event types:

- user — Login and authentication related events

Entity Management contains the following event types:

- user — User creation and deletion
- content
- comment

HTTP Activity contains the following event types:

- access denied

- page not found
- php
- new custom types

Application Lifecycle contains the following event types:

- system
- cron

CloudWatch pipelines configuration for Drupal Core

Collects audit logs from Drupal Core using basic authentication.

Configure the Drupal source with the following parameters:

```
source:
  drupal_core:
    range: "P7D"
    domain: "<drupal-site-domain>"
    api_endpoint: "<drupal-api-endpoint>"
    authentication:
      basic:
        username: "${{aws_secrets:<secret-name>:username}}"
        password: "${{aws_secrets:<secret-name>:password}}"
```

Parameters

`range` (optional)

For pulling historical logs. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for the last 21 hours). The default is 0 hours, and the maximum is 90 days.

`domain` (required)

The domain name of the Drupal Core site.

`api_endpoint` (required)

The path of the Drupal Core audit logs API endpoint. Must start with `/` (for example, `/api/v1/audit-logs`).

`authentication.basic.username` (required)

The username for Basic Authentication.

`authentication.basic.password` (required)

The password for Basic Authentication.

Entrust IDaaS integration configuration

Entrust Identity as a Service (IDaaS) is a cloud-based Identity and Access Management (IAM) platform that provides multi-factor authentication (MFA), single sign-on (SSO), adaptive risk-based authentication, and comprehensive audit logging across workforce, consumer, and citizen use cases.

Topics

- Source configuration for Entrust IDaaS
- CloudWatch pipelines configuration for Entrust IDaaS

Source configuration for Entrust IDaaS

Integrating with Entrust IDaaS

Entrust Identity as a Service (IDaaS) is a cloud-based Identity and Access Management (IAM) platform that provides multi-factor authentication (MFA), single sign-on (SSO), adaptive risk-based authentication, and comprehensive audit logging across workforce, consumer, and citizen use cases. CloudWatch pipeline uses the Entrust IDaaS Administration REST API to retrieve identity and access events from your IDaaS tenant.

The Administration REST API provides access to two primary log categories. Authentication Logs capture user authentication events across multiple event types, including MFA, SSO, SAML, OIDC, and passwordless authentication methods. Management Logs track administrative actions and changes performed across various entity types such as users, groups, applications, tokens, and policies.

Authenticating with Entrust IDaaS

To read the logs, the pipeline needs to authenticate with your Entrust IDaaS tenant. The plugin supports Administration API authentication using an `applicationId` and `sharedSecret`.

Create an Administration API application

- Go to your IDaaS Admin portal and navigate to Security → Applications.
- Choose + and select Administration API from the list of available applications.
- In the General tab, enter a name and description for your application, then choose Next.
- In the Setup tab, assign the role with the permissions required by your application, then choose Submit. The Entrust IDaaS Administration API requires the Super Administrator role to access audit log endpoints.
- In the Complete tab, choose Copy to copy your `applicationId` and `sharedSecret`, or download the JSON file.
- In AWS Secrets Manager, create a secret and store the `applicationId` under the key `client_id` and the `sharedSecret` under the key `client_secret`.
- Your IDaaS API base URL is `https://<hostname>` where `hostname` is taken from the credentials (for example, `https://entrust.us.trustedauth.com`).

Configuring the CloudWatch Pipeline

To configure the pipeline to read audit logs from Entrust IDaaS, choose `entrust_idaas` as the data source. Fill in the required information such as your tenant `hostname` and the AWS Secrets Manager secret ARN for your credentials where `client_id` and `client_secret` are stored. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to Authentication (3002) and Entity Management (3004).

Authentication contains the following events:

- AuthenticationAdminApiSuccessEvent
- AuthenticationDeniedEvent
- AuthenticationExternalSecondFactorBypassEvent
- AuthenticationExternalSuccessEvent
- AuthenticationFaceSuccessEvent
- AuthenticationFidoSuccessEvent
- AuthenticationFirstFactorExternalSuccessEvent

- AuthenticationFirstFactorIdpSuccessEvent
- AuthenticationFirstFactorPasswordSuccessEvent
- AuthenticationGridSuccessEvent
- AuthenticationGridWithTempAccessCodeSuccessEvent
- AuthenticationIdpSuccessEvent
- AuthenticationKbaSuccessEvent
- AuthenticationLockedEvent
- AuthenticationMagicLinkSuccessEvent
- AuthenticationOtpCreatedEvent
- AuthenticationOtpEmailSentEvent
- AuthenticationOtpNoCreditEvent
- AuthenticationOtpSentToAllEvent
- AuthenticationOtpSmsSentEvent
- AuthenticationOtpSuccessEvent
- AuthenticationOtpUnavailableEvent
- AuthenticationOtpVoiceSentEvent
- AuthenticationOtpWithTempAccessCodeSuccessEvent
- AuthenticationPasskeySuccessEvent
- AuthenticationPasswordSuccessEvent
- AuthenticationSecondFactorFaceSuccessEvent
- AuthenticationSecondFactorFIDOSuccessEvent
- AuthenticationSecondFactorGridSuccessEvent
- AuthenticationSecondFactorGridWithTempAccessCodeSuccessEvent
- AuthenticationSecondFactorKbaSuccessEvent
- AuthenticationSecondFactorMagicLinkSuccessEvent
- AuthenticationSecondFactorOtpSuccessEvent
- AuthenticationSecondFactorOtpWithTempAccessCodeSuccessEvent
- AuthenticationSecondFactorSmartCredentialPushSuccessEvent
- AuthenticationSecondFactorTempAccessCodeSuccessEvent

- AuthenticationSecondFactorTokenSuccessEvent
- AuthenticationSecondFactorTokenWithTempAccessCodeSuccessEvent
- AuthenticationSecondFactorUserCertificateSuccessEvent
- AuthenticationSmartCredentialPushSuccessEvent
- AuthenticationSmartLoginSuccessEvent
- AuthenticationTempAccessCodeSuccessEvent
- AuthenticationTokenPushSuccessEvent
- AuthenticationTokenSuccessEvent
- AuthenticationTokenWithTempAccessCodeSuccessEvent
- AuthenticationUserCertificateSuccessEvent
- MachineLockedEvent
- OidcAuthenticationFailedEvent
- OidcAuthenticationSuccessEvent
- SamlAuthenticationFailedEvent
- SamlAuthenticationSuccessEvent
- UserPasswordChangeFailedEvent
- UserPasswordChangeLockedEvent
- UserStepUpAuthenticationSuccessEvent
- VerificationDeniedEvent
- VerificationIdpSuccessEvent

Entity Management contains the following events:

- ACTIVESYNC
- AD_CONNECTOR_DIRECTORIES
- AGENTS
- APPLICATIONS
- ARCHIVES
- AUTHENTICATIONFLOWS
- AUTHORIZATIONGROUPS

- AZURE_DIRECTORIES
- BLACKLISTEDPASSWORDS
- BULKENROLLMENTS
- BULKGROUPS
- BULKHARDWARETOKENS
- BULKIDENTITYGUARD
- BULKSMARTCARDS
- BULKUSERS
- CAS
- CERTIFICATES
- CLAIMS
- CONTACTVERIFICATION
- CONTEXTRULES
- CREATETENANT
- CREDENTIALDESIGNS
- CUSTOMIZATIONVARIABLES
- DIGITALIDCERTIFICATES
- DIGITALIDCONFIGCERTTEMPS
- DIGITALIDCONFIGS
- DIGITALIDCONFIGSANS
- DIGITALIDCONFIGVARIABLES
- DIRECTORIES
- DIRECTORYATTRIBUTES
- DIRECTORYCONNECTIONS
- DIRECTORYPASSWORD
- DIRECTORYSEARCHATTRIBUTES
- DIRECTORYSYNC
- DOMAINCONTROLLERCERTS
- EMAILTEMPLATES

- EMAILVARIABLES
- ENROLLMENTDESIGNS
- ENROLLMENTS
- ENTITLEMENTS
- EXPECTEDLOCATIONS
- EXPORTREPORTS
- FACE
- FIDOTOKENS
- GATEWAYCSRS
- GATEWAYS
- GRIDCONTENTS
- GRIDS
- GROUPPOLICIES
- GROUPS
- HIGH_AVAILABILITY_GROUPS
- HOSTNAMESETTINGS
- IDENTITYPROVIDERS
- IDPROOFING
- IDPROOFINGLICENSE
- INTELLITRUSTDESKTOPS
- IPLISTS
- ISSUANCE
- MAGICLINKCONTENTS
- MAGICLINKS
- OAUTHROLES
- ORGANIZATIONS
- OTPPROVIDERS
- OTPS
- PIVCONTENTSIGNER

- PKIAASCREDENTIALS
- POLICYOVERRIDE
- PREFERREDOTPPROVIDERS
- PRINTERS
- PUSHCREDENTIALS
- QUESTIONS
- RATELIMITING
- REPORTS
- RESOURCESERVERAPIS
- RESOURCESERVERSCOPES
- RISKENGINES
- ROLES
- SCDEFNPIVAPPLETCONFIGS
- SCDEFNS
- SCDEFNVARIABLES
- SCHEDULEDTASKS
- SCIMPROVISIONINGS
- SENDAZUREAD
- SENDEMAIL
- SENDSCIM
- SERVICEPROVIDERACCOUNTS
- SERVICEPROVIDERS
- SETTINGS
- SMARTCARDS
- SMARTCREDENTIALS
- SMARTCREDENTIALSSIGNATURE
- SPCLIENTCREDENTIALS
- SPENTITLEMENTS
- SPIDENTITYPROVIDERS

- SPMANAGEMENTPLATFORM
- SPROLES
- SPUSERMGMT
- SUBSCRIBERS
- TEMPACCESSCODECONTENTS
- TEMPACCESSCODES
- TEMPLATES
- TENANTS
- TOKENACTIVATIONCONTENTS
- TOKENS
- TRANSACTIONITEMS
- TRANSACTIONRULES
- USERATTRIBUTES
- USERATTRIBUTEVALUES
- USERKBACHALLENGES
- USERLOCATIONS
- USERMACHINES
- USEROAUTHTOKENS
- USERPASSWORDS
- USERQUESTIONANSWERS
- USERQUESTIONS
- USERRBASETTINGS
- USERS
- USERSITEROLES
- USERSPROLES
- WORDSYNONYMS

CloudWatch pipelines configuration for Entrust IDaaS

Collects identity and access management audit logs from Entrust IDaaS using OAuth2 authentication.

Configure the Entrust IDaaS source with the following parameters:

```
source:
  entrust_idaas:
    hostname: "<hostname>"
    range: "P7D"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`hostname` (required)

Entrust IDaaS tenant hostname (for example, `entrust.us.trustedauth.com`). Do not include the `https://` prefix.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Entrust IDaaS Administration API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Entrust IDaaS Administration API authentication.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

> **Note**
>
> The parameter information should correspond to values received in the Authenticating with Entrust IDaaS section.

F5 BIG-IP integration configuration

F5 Telemetry Streaming is a real-time log streaming solution that delivers detailed traffic and security logs from F5 BIG-IP to external destinations such as Amazon S3 and CloudWatch. It

enables monitoring, troubleshooting, and analysis of logs generated by modules like AFM, LTM, APM, ASM, and Advanced WAF.

Topics

- Source configuration for F5 BIG-IP
- CloudWatch pipelines configuration for F5 BIG-IP

Source configuration for F5 BIG-IP

Integrating with F5 BIG-IP

To integrate F5 BIG-IP with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your F5 BIG-IP source by configuring Amazon S3 and Amazon SQS to receive data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Setup log forwarding

F5 BIG-IP supports real-time log delivery through Telemetry Streaming to forward logs to Amazon S3.

Instructions to setup Amazon S3 and Amazon SQS

Configuring F5 BIG-IP to send logs to an Amazon S3 bucket involves several steps. These steps focus on setting up the Amazon S3 bucket, Amazon SQS queue, and IAM credentials, and then configuring Telemetry Streaming and the CloudWatch pipeline.

- Make sure F5 BIG-IP Telemetry Streaming is installed and configured. Create an Amazon S3 bucket to store logs. It is recommended to enable server-side encryption for security.
- Amazon S3 bucket that stores the F5 BIG-IP logs should reside in the same AWS Region.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring F5 BIG-IP log forwarding

- Install and verify the Telemetry Streaming (TS) extension.

- Create a log destination (Amazon S3, CloudWatch, or other supported destination).
- Create a log publisher and map the destination.
- Create a logging profile specific to each module (for example, ASM, Advanced WAF).
- Apply the logging profile to the virtual server.
- Configure Telemetry Streaming (JSON declaration).
- Define the destination S3 bucket and log types.

Configuring Telemetry Streaming

- Configure Telemetry Streaming using the declarative API endpoint: `/mgmt/shared/telemetry/declare`.
- Define the Telemetry configuration with class `Telemetry`.
- Define a Telemetry Listener to receive logs (for example, port 6514).
- Define a Telemetry Consumer of type Amazon S3: configure the bucket name and Region, and provide IAM access key and secret key for authentication.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from F5 BIG-IP, choose F5 BIG-IP as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and F5 BIG-IP events that map to Network Activity (4001) and HTTP Activity (4002). The following lists show the source for each event.

Network Activity (4001) contains the following log formats:

- AFM
- APM

HTTP Activity (4002) contains the following log formats:

- ASM

- LTM
- Advanced WAF

Events that do not match any OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

CloudWatch pipelines configuration for F5 BIG-IP

The F5 BIG-IP setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the F5 BIG-IP source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "f5_bigip"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "f5_bigip".

`aws.region` (required)

The AWS Region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

GitHub Audit Log integration configuration

Amazon Telemetry Pipelines enables you to collect audit logs from GitHub Enterprise Cloud. GitHub Enterprise is an enterprise-grade software development platform designed for the complex workflows of modern development.

Topics

- Source configuration for GitHub Audit Log
- CloudWatch pipelines configuration for GitHub Audit Log

Source configuration for GitHub Audit Log

> **Note**
>
> Important: GitHub Enterprise accounts are required to use this connector. GitHub Personal or Organization accounts are not supported.

Integrating with GitHub

Amazon Telemetry Pipelines enables you to collect audit logs from GitHub Enterprise Cloud. GitHub Enterprise is an enterprise-grade software development platform designed for the complex workflows of modern development. GitHub Enterprise Cloud is the cloud-based solution of GitHub Enterprise, hosted on GitHub's servers.

Authenticating with GitHub

To read the audit logs, pipeline needs to authenticate with your GitHub account. For Enterprise scope , you can use Personal Access Token and for Organization scope , you can either use Personal Access Token or GitHub App.

Generate the token to authenticate as Personal Access Token:

- Sign in to GitHub using credentials for the GitHub account
- The authenticated user must be an enterprise admin to use this endpoint
- Open the GitHub Personal access tokens (classic) page, locate the Generate new token (classic) and then follow the GitHub procedure to generate a token with `read:audit_log` scope and No expiration
- Store this new token in a secret in the AWS Secrets Manager under the key `personal_access_token`

Generate the private key to authenticate as GitHub App:

- Sign in to GitHub using credentials for the GitHub account
- Make sure the GitHub App has the "Administration" organization permissions (read) permission
- Follow the instructions in Managing private keys for GitHub Apps and generate the private key
- Store this private key in a secret in the AWS Secrets Manager under the key `private_key` and the GitHub App name under the key `app_id`

Configuring the CloudWatch Pipeline

When configuring the pipeline to read audit logs from GitHub Enterprise Cloud, choose GitHub Audit Logs as the data source. Select the Source Type as Enterprise or Organization based on the scope of your integration and fill in the required information like Enterprise or Organization name

according to the selected scope. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and the GitHub actions that maps to Account Change (3001), API Activity (6003) and Entity Management (3004).

Account Change contains the following actions:

- org.enable_two_factor_requirement
- org.disable_two_factor_requirement
- two_factor_authentication.add_factor
- two_factor_authentication.enabled
- two_factor_authentication.disabled
- two_factor_authentication.remove_factor
- org.disable_saml
- org.enable_saml
- personal_access_token.access_restriction_disabled
- personal_access_token.access_restriction_enabled
- personal_access_token.expiration_limit_set
- personal_access_token.expiration_limit_unset

API Activity contains the following actions:

- repository_secret_scanning_custom_pa....create
- repository_secret_scanning_custom_pa....update
- repository_secret_scanning_custom_pa....delete
- repository_secret_scanning_custom_pa....publish
- repository_secret_scanning_custom_p....enabled
- repository_secret_scanning_custom_p....disabled
- repository_secret_scanning_non_provi....enabled
- repository_secret_scanning_non_provi....disabled

- repository_secret_scanning_generic_s....enabled
- repository_secret_scanning_generic_s....disabled
- business_secret_scanning_custom_pattern.create
- business_secret_scanning_custom_pattern.update
- business_secret_scanning_custom_pattern.delete
- business_secret_scanning_custom_pattern.publish
- business_secret_scanning_custom_patt....enabled
- business_secret_scanning_custom_patt....disabled
- business_secret_scanning_generic_secrets.enabled
- business_secret_scanning_generic_secrets.disabled
- business_secret_scanning_non_provide....enabled
- business_secret_scanning_non_provide....disabled
- org_secret_scanning_non_provider_patt....enabled
- org_secret_scanning_non_provider_patt....disabled
- org_secret_scanning_generic_secrets.enabled
- org_secret_scanning_generic_secrets.disabled
- org_secret_scanning_custom_pattern.create
- org_secret_scanning_custom_pattern.update
- org_secret_scanning_custom_pattern.delete
- org_secret_scanning_custom_pattern.publish

Entity Management contains the following actions:

- oauth_application.destroy
- oauth_application.generate_client_secret
- oauth_application.remove_client_secret
- oauth_application.revoke_all_tokens
- oauth_application.revoke_tokens
- oauth_application.transfer
- personal_access_token.auto_approve_grant_requests_enabled

- personal_access_token.auto_approve_grant_requests_disabled
- ip_allow_list.disable
- ip_allow_list.enable_for_installed_apps
- ip_allow_list.disable_for_installed_apps
- ip_allow_list_entry.create
- ip_allow_list_entry.update
- ip_allow_list_entry.destroy
- repository_secret_scanning.disable
- repository_secret_scanning_automatic....disabled
- repository_secret_scanning_push_prot....disable
- repository_secret_scanning_push_prot....enable
- oauth_application.create
- oauth_application.reset_secret
- auto_approve_personal_access_token_req....enabled
- auto_approve_personal_access_token_req....disabled
- ip_allow_list.enable
- ip_allow_list.disable_user_level_enforcement
- ip_allow_list.enable_user_level_enforcement
- repository_secret_scanning.enable
- repository_secret_scanning_automatic....enabled
- repository_secret_scanning_push_prot....enable
- repository_secret_scanning_push_prot....add
- repository_secret_scanning_push_prot....remove
- repository_secret_scanning_push_prot....disable
- secret_scanning.enable
- secret_scanning.disable
- secret_scanning_new_repos.enable
- org_secret_scanning_automatic_validi....enabled
- org_secret_scanning_automatic_validi....disabled

- org_secret_scanning_push_protection_b....add
- org_secret_scanning_push_protection_b....remove
- org_secret_scanning_push_protection_b....disable
- org_secret_scanning_push_protection_b....enable
- business_secret_scanning_automatic_va....enabled
- business_secret_scanning_automatic_va....disabled
- business_secret_scanning_push_protection.enable
- business_secret_scanning_push_protection.disable
- business_secret_scanning_push_protection.enabled_for_new_repos
- business_secret_scanning_push_protection.disabled_for_new_repos
- business_secret_scanning_push_prote....enable
- business_secret_scanning_push_prote....update
- business_secret_scanning_push_prote....disable

CloudWatch pipelines configuration for GitHub Audit Log

> **Note**
>
> Important: GitHub Enterprise accounts are required to use this connector. GitHub Personal or Organization accounts are not supported.

Collects audit log data from GitHub organizations or enterprises using personal access tokens or GitHub App authentication.

Configure the GitHub Audit Log source with the following parameters:

```
source:
  github_auditlog:
    scope: "ORGANIZATION"
    organization: "<example-org-name>"
    range: "P7D"
    authentication:
      personal_access_token: "${{aws_secrets:<secret-name>:token}}"
```

Parameters

`scope` (required)

Scope of audit logs to collect. Must be "ORGANIZATION" or "ENTERPRISE".

`organization` (required when scope is ORGANIZATION)

GitHub organization name.

`enterprise` (required when scope is ENTERPRISE)

GitHub enterprise name.

`authentication.personal_access_token` (required for PAT auth)

Personal access token for GitHub API authentication.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

GitLab integration configuration

GitLab is a DevSecOps platform that provides source code management, CI/CD pipelines, project planning, and security scanning capabilities. CloudWatch Pipelines use the GitLab REST API and GraphQL API to retrieve audit events, project activity, merge request data, group membership information, and vulnerability findings from your GitLab environment.

Topics

- Source configuration for GitLab
- CloudWatch pipelines configuration for GitLab

Source configuration for GitLab

Integrating with GitLab

CloudWatch Pipeline uses the GitLab REST API and GraphQL API to retrieve audit events, project activity, merge request data, group membership information, and vulnerability findings from your GitLab environment. The REST API exposes endpoints for fetching time-filtered event data for security monitoring and compliance analysis, while the GraphQL API is used to retrieve project vulnerability data.

To integrate CloudWatch Pipelines with GitLab, complete the following high-level steps:

- Generate a Personal Access Token (PAT) in the GitLab Console.
- Store the credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with GitLab as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- An active GitLab (SaaS) account with Owner or Admin access
- A GitLab Personal Access Token (PAT) with the `read_api` scope
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager

Authenticating with GitLab

GitLab uses a Personal Access Token (PAT) for authentication. The pipeline passes the token through the `PRIVATE-TOKEN` header to authenticate each API request.

Configure Authentication for GitLab

To configure authentication credentials for the pipeline:

1. In the upper-right corner of GitLab, select your avatar.
2. Select **Edit profile**.
3. In the left sidebar, select **Access** > **Personal access tokens**.
4. From the **Generate token** dropdown, select **Legacy token**.
5. Enter a token name.
6. (Optional) Enter a token description.
7. Set an expiration date for the token.
8. Select `read_api` as the scope.
9. Select **Generate token**.
10. Save the token value.

11Store the `pat_token` in AWS Secrets Manager.

Configuring the CloudWatch Pipeline

To configure the pipeline, choose GitLab as the data source. Provide the Personal Access Token. After you create and activate the pipeline, audit and activity data from GitLab will begin flowing into the selected CloudWatch Logs log group.

The following optional parameters are available:

- **Polling intervals** – `members_polling_interval` (default `P1D`), `vulnerabilities_polling_interval` (default `P1D`)
- **Backfill durations** – `projects_backfill` (default `P180D`), `merge_requests_backfill` (default `P180D`), `group_audit_events_backfill` (default `P180D`), `project_audit_events_backfill` (default `P180D`)

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports the following OCSF event classes:

- **Vulnerability Finding [2002]** – Vulnerability Finding Activity
- **Group Management [3006]** – Group Audit Event Activity
- **User Inventory Info [5003]** – Member Activity
- **Software Inventory Info [5020]** – Project Activity
- **Web Resources Activity [6001]** – Merge Request Activity

CloudWatch pipelines configuration for GitLab

Collects log data from GitLab using Personal Access Token authentication through the GitLab REST API and GraphQL API.

Configure the GitLab source with the following parameters:

```
source:
  gitlab:
    authentication:
      personal_access_token: "${{aws_secrets:gitlab-account-credentials:pat_token}}"
    range: "P30D"
```

Parameters

`authentication.personal_access_token` (required)

The GitLab Personal Access Token with `read_api` scope, stored in AWS Secrets Manager.

`range` (optional)

The historical time period for backfilling data. Uses ISO 8601 duration format. Minimum is `PT1H`, maximum is `P180D`. Default is `P180D`.

Google Workspace integration configuration

Google Workspace, formerly G Suite, is Google's cloud-native productivity and collaboration suite. It includes Gmail, Drive, Docs, Sheets, Slides, Calendar, Meet, and Chat, along with identity and security services. CloudWatch pipelines use the Google Workspace Admin SDK Reports API and Alert Center API. These APIs retrieve audit logs, authentication activity, account and access changes, drive operations, and security alerts. The data comes from your Google Workspace environment. Both APIs expose REST endpoints that fetch time-filtered event data for security monitoring and analysis.

Topics

- Source configuration for Google Workspace
- CloudWatch pipelines configuration for Google Workspace

Source configuration for Google Workspace

Supported API and platform versions

Component	Version	Notes
Admin SDK Reports API	v1	Activity reports for all supported applications
Alert Center API	v1beta1	Security alerts and investigation data
OCSF Schema	v1.5.0	Open Cybersecurity Schema Framework mapping
OAuth 2.0	2	Service account authentication with a JSON Web Token (JWT)

Prerequisites

Before you begin, make sure you have the following:

- An active Google Workspace subscription. Individual applications and reports can require specific Google Workspace editions. For more information, see the section called “Known platform limitations”.
- A Google Cloud project.
- A Google Cloud service account with domain-wide delegation enabled.
- An administrator email address available for impersonation.
- The Admin SDK Reports API enabled in the Google Cloud console.
- The Alert Center API enabled in the Google Cloud console.
- Super-admin access to the Google Admin console for domain-wide delegation authorization.
- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called “API caller permissions”.
- An AWS account with permissions to create and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called “Third-party sources (API Pull)”.
- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called “CloudWatch Logs API operations and required permissions for actions”.
- The service account JSON key file that you create during Google Workspace configuration.

Configure Google Workspace

1. Navigate to the Google Cloud console, log in to your Google account, and choose your organization name in the navigation bar.
2. In the **Select a resource** window, choose **New Project**.
3. In the **New Project** window, enter your project name, and choose **Create**.
4. In the Google Cloud console, open the menu in the upper-left corner.
5. Navigate to **APIs & Services** > **Library**.
6. Choose the search bar.
7. Search for and select **Admin SDK API**. Repeat this step for **Google Workspace Alert Center API**.

8. For both the Admin SDK API and Google Workspace Alert Center API, choose **Enable**.
9. Navigate to **IAM & Admin** > **Service Accounts**.
10. Choose **Create Service Account**.
11. Name your service account, choose **Create and Continue**, and then choose **Done**.
12. On the **Service Accounts** page, choose your new service account.
13. On the service account details page, create a JSON key:
    a. Choose the **Keys** tab, and then choose **Add Key**.
    b. Choose **Create new key**.
    c. Select **JSON**, and then choose **Create**.
    d. Save the JSON key file to a secure directory. You use the credentials from this file when you configure the pipeline.
14. Navigate to the Google Admin console, and then navigate to **Security** > **Access and data control** > **API controls**.
15. Under **Domain wide delegation**, choose **Manage Domain Wide Delegation**.
16. Choose **Add new** to add a new client ID.
17. In the **Add a new client ID** window, complete the following steps:
    a. For **Client ID**, enter the value of the `client_id` key from the service account JSON key file.
    b. For **OAuth scopes**, add the following comma-delimited scopes:
       - Admin Audit Read Only
       - Alerts
    c. Choose **Authorize**.

Integrating with Google Workspace

1. **Store credentials in AWS Secrets Manager**
    a. Open the AWS Secrets Manager console.
    b. Choose **Store a new secret** > **Other type of secret**.
    c. Add the following key-value pairs:
       - `private_key` – The `private_key` value from the service account JSON key file.
       - `client_email` – The `client_email` value from the service account JSON key file.
       - `subject` – The administrator email address to impersonate.

d. Name the secret, for example, `google-workspace/pipeline-credentials`.

2. **Create the destination CloudWatch Logs log group**

   a. Open the CloudWatch console.

   b. Navigate to **Logs** > **Log groups** > **Create log group**.

   c. Choose a descriptive name, for example, `/aws/cloudwatch/pipelines/google-workspace`.

3. **Create the source IAM role**

   Create an IAM role with the required permissions. For information about the required policies and permissions, see the section called “CloudWatch pipelines IAM policies and permissions”.

4. **Create the CloudWatch pipeline**

   a. Open the CloudWatch console, and navigate to **Pipelines**.

   b. Choose **Google Workspace** as the data source.

   c. Configure authentication with `client_email`, `private_key` from AWS Secrets Manager, and `subject`, which is the administrator email address to impersonate.

   d. Select the destination log group.

   For the complete pipeline YAML configuration and parameter reference, see the section called “Pipeline configuration”.

5. **Add the CloudWatch Logs resource policy**

   The resource policy behavior depends on whether the log group is new or already exists:

   - **New log group (first-time use)** – The resource policy is created automatically. No action is required.
   - **Existing log group (reusing)** – If the selected log group already has a resource policy, the console displays the following warning:

     Resource Policy Detected: The log group you selected already has a resource policy. Please verify that this policy includes permissions for pipelines to write to the destination log group. You may need to modify the existing resource policy to add the necessary permissions.

     In this case, add the CloudWatch Logs resource policy within 5 minutes of pipeline creation, before the pipeline becomes active. Review and update the existing resource policy to make sure that it includes the required pipeline write permissions.

> **Note**
>
> If you create the pipeline using the AWS CLI or API, a resource policy is never created automatically, regardless of whether the log group is new or existing. You must create the CloudWatch Logs resource policy manually before the pipeline becomes active.

6. **Verify data flow**
   a. Wait a few minutes for the initial data pull.
   b. Check the destination log group for incoming events.
   c. Monitor pipeline metrics for successful ingestion.

Authenticating with Google Workspace

Google Workspace uses OAuth 2.0 with service account authentication and domain-wide delegation. The pipeline performs the following actions:

1. Signs a JWT with the service account private key.
2. Exchanges the JWT for an access token through Google's OAuth endpoint.
3. Impersonates the specified administrator account to access organization-wide data.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0. Each application maps to one or more OCSF event classes.

Event type	Application ID	OCSF class	Description
Login Activity	`login`	Authentication (3002), Account Change (3001), Detection Finding (2004), Base Event (0)	User authentication attempts, MFA challenges, login success or failure, suspicious login flags, and password changes
SAML Activity	`saml`	Authentication (3002)	SAML-based SSO authentication attempts for federated applicati ons

Event type	Application ID	OCSF class	Description
Token Activity	`token`	API Activity (6003), User Access Management (3005)	OAuth token grants, revocations, and third-party application access
Drive Activity	`drive`	File Hosting Activity (6006)	File sharing, downloads, deletions, and permission changes
Gmail Activity	`gmail`	Email Activity (4009)	Message delivery events, including messages sent and received
Context-Aware Access	`context_aware_access`	Detection Finding (2004)	Zero-trust policy enforcement events, including access denials, warn-mode events, and policy evaluation errors
Rules Activity	`rules`	Detection Finding (2004), Data Security Finding (2006)	DLP and activity rules triggered by Google's built-in rule engine
Access Transparency	`access_transparency`	API Activity (6003)	Google employee access to customer data, available with the Frontline Plus, Enterprise Plus, Education Standard, Education Plus, and Enterprise Essentials Plus editions
User Accounts Activity	`user_accounts`	Account Change (3001)	2-step verification enrollment or disablement and recovery information changes
LDAP Activity	`ldap`	API Activity (6003), Authentication (3002)	LDAP authentication operations against Secure LDAP
Takeout Activity	`takeout`	Web Resources Activity (6001)	Bulk data export and download operations through Google Takeout

Event type	Application ID	OCSF class	Description
Vault Activity	`vault`	Entity Management (3004)	eDiscovery and retention operations for compliance monitoring
Directory Sync Activity	`directory_sync`	Entity Management (3004)	Active Directory and LDAP directory synchronization operations
Alert Center	Not applicable	Detection Finding (2004)	Security alerts, including phishing reclassifications, government-backed attack warnings, leaked passwords, and suspicious sharing

Ingestion-only event types

The following event types are ingested for completeness but do not have OCSF class mappings:

Event type	Application ID	Description
Calendar Activity	`calendar`	Calendar ACL changes and event modifications
Chat Activity	`chat`	Space management, uploads, and message operations
Meet Activity	`meet`	Video call events and abuse reports
Chrome Activity	`chrome`	ChromeOS logins, unsafe browsing events, and device management

CloudWatch pipelines configuration for Google Workspace

Collects audit logs from Google Workspace using OAuth 2.0 service account authentication with domain-wide delegation.

YAML configuration

```
extension:
  aws:
    secrets:
      google_workspace:
        secret_id: <<arn:aws:secretsmanager:<region>:<account-id>:secret:<secret-
name>>>
        region: <<region>>
        sts_role_arn: <<arn:aws:iam::<account-id>:role/<role-name>>>
pipeline:
  source:
    google_workspace:
      user_key: all
      range: P30D
      authentication:
        jwt_key_pair:
          private_key: ${{aws_secrets:google_workspace:private_key}}
          client_email: ${{aws_secrets:google_workspace:client_email}}
          subject: ${{aws_secrets:google_workspace:subject}}
  sink:
    - cloudwatch_logs:
        log_group: <<google-workspace-log-group>>
  processor:
    - ocsf:
        schema:
          google_workspace: null
        version: "1.5"
        mapping_version: 1.5.0
```

AWS extension parameters

The AWS extension configures AWS Secrets Manager access for securely retrieving credentials at runtime. Define a named secret block, such as `google_workspace`, with `secret_id` set to the ARN of your AWS Secrets Manager secret, `region` set to the AWS Region where the secret is stored, and `sts_role_arn` set to the IAM role ARN to assume when accessing the secret. Credentials are then referenced in the pipeline by using `${{aws_secrets:<name>:<key>}}`. For the complete configuration reference, see the AWS Secrets Manager extension.

Source parameters

Parameters

`user_key` (optional)

String that provides the profile ID or user email for which the data is filtered. The default value, `all`, fetches all information.

`range` (optional)

Lookback duration for activity logs in ISO 8601 format. The minimum value is `P1D`, the maximum value is `P180D`, and the default value is `P180D`.

`authentication.jwt_key_pair.private_key` (required)

The private key from the Google Cloud service account JSON key file. Store the value in AWS Secrets Manager and reference it using `${{aws_secrets:google_workspace:private_key}}`.

`authentication.jwt_key_pair.client_email` (required)

The service account email address. Store the value in AWS Secrets Manager and reference it using `${{aws_secrets:google_workspace:client_email}}`.

`authentication.jwt_key_pair.subject` (required)

The administrator email address to impersonate through domain-wide delegation. This value must be a delegated administrator account, not the service account email. Store the value in AWS Secrets Manager and reference it using `${{aws_secrets:google_workspace:subject}}`.

Note

Store all three authentication values, `private_key`, `client_email`, and `subject`, in AWS Secrets Manager. Reference them through the `google_workspace` secret block. The service account credentials, `private_key` and `client_email`, are available in the Google Cloud console under **IAM & Admin** > **Service Accounts**. The `subject`, or delegated administrator email, must be an administrator in your Google Workspace organization with domain-wide delegation authorized for the service account.

> **Note**
>
> For Gmail activity, if `range` is set to more than `P30D`, only the last 30 days of logs are fetched because of API constraints.

Sink parameters

The sink delivers processed events to a CloudWatch Logs log group. Set `log_group` to the name of your destination log group, for example, `google-workspace-log-group`. For complete configuration options, see the CloudWatch Logs sink.

Processor parameters

The processor transforms ingested events into OCSF format using the `google_workspace` schema key. Set `schema.google_workspace` to `null`. The presence of the key activates the Google Workspace schema mapping. Set `version` to `"1.5"` and `mapping_version` to `1.5.0`. For the full parameter reference, see the OCSF processor.

Known platform limitations

Limitation	Details	Impact
Gmail query window	The Reports API requires a start time and an end time for every Gmail request, and the difference between them cannot be greater than 30 days.	`range` values exceeding `P30D` fetch only the most recent 30 days of Gmail data.
Activity data availability delay	The delay before events become available in the Reports API depends on the application, and ranges from near real time to several days.	For the slower applications, recent events might not appear in the destination log group immediately.
Alert Center availability	The Alert Center API is in beta, version v1beta1.	API behavior can change without notice.

Limitation	Details	Impact
Access Transparency edition requirements	Access Transparency logs are available only with the Frontline Plus, Enterprise Plus, Education Standard, Education Plus, and Enterprise Essentials Plus editions.	If your organization uses an edition that is not listed, you do not receive these events.
Domain-wide delegation	Domain-wide delegation requires super-admin privileges to configure.	If you are not a super-admin, you cannot configure the integration.
API rate limits	The Reports API allows 2,400 queries per minute for each user in a Google Cloud project. The Alert Center API allows 1,000 requests per second for each project and 150 requests per second for each user.	High-volume environments might experience throttling.

Troubleshooting

Error	Likely cause	Resolution or connector behavior
`401 Unauthorized`	Invalid or expired service account credentials	Verify that the `private_key` stored in AWS Secrets Manager matches the current service account key. If necessary, regenerate the key in the Google Cloud console.
`403 Forbidden – Not Authorized to access this resource/ api`	Domain-wide delegation is not configured or scopes are not authorized.	Verify that the service account client ID is added in the Google Admin console with the correct OAuth scopes. Make sure that the `subject` is an administrator account.

Error	Likely cause	Resolution or connector behavior
`403 Forbidden – Access Not Configured`	A required API is not enabled.	Enable the Admin SDK API and Alert Center API in the Google Cloud console.
`429 Too Many Requests`	An API rate limit is exceeded.	The connector automatically backs off and retries. If the error persists, verify that other workloads are not consuming the same Google Workspace API quota.
`400 Bad Request – Invalid subject`	The subject email is not a valid administrator account.	Verify that the `subject` value is an administrator email address in your Google Workspace organization.
No data appears in the log group.	The pipeline resource policy was not added within 5 minutes of pipeline creation, before the pipeline became active.	Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.
Some applications are missing from the data.	Service account scopes are too restrictive.	Verify that all required OAuth scopes are authorized in the Google Admin console domain-wide delegation settings.

Gmail activity shows fewer events than expected

- The Reports API limits each Gmail request to a window of 30 days, so only the most recent 30 days of Gmail activity are fetched regardless of the configured `range` value.
- Gmail activity reports message delivery events, such as messages sent and received.

Alert Center returns empty results

- The current version of the Alert Center API, v1beta1, is available to all Google Workspace customers. Access depends on a privilege rather than on an edition. Verify that the impersonated `subject` account holds the Alert center administrator privilege.
- Alerts can take up to 24 hours to appear after the triggering event.
- Verify that the `https://www.googleapis.com/auth/apps.alerts` scope is authorized.

Access Transparency logs do not appear

- This feature requires the Frontline Plus, Enterprise Plus, Education Standard, Education Plus, or Enterprise Essentials Plus edition.
- Verify that your organization opted in to Access Transparency in the Google Admin console.

HashiCorp Vault integration configuration

HashiCorp Vault is the industry-leading secrets management and encryption-as-a-service platform that enables organizations to centrally store, access, and distribute digital secrets. CloudWatch Pipelines allow ingestion of HashiCorp Vault audit log and operational log data into CloudWatch Logs.

Topics

- Source configuration for HashiCorp Vault
- CloudWatch pipelines configuration for HashiCorp Vault

Source configuration for HashiCorp Vault

Integrating with HashiCorp Vault

HashiCorp Vault integration uses Amazon S3 and Amazon SQS to ingest log data into CloudWatch pipelines. Vault produces two log types: audit logs (detailed JSON records of all API requests and responses) and operational logs (server health and performance). Fluent Bit collects both log types from the Vault host and delivers them to an Amazon S3 bucket. Amazon SQS notifications alert the pipeline when new log objects arrive.

Prerequisites

- An AWS account with permissions to create Amazon S3 buckets, Amazon SQS queues, and IAM roles
- A HashiCorp Vault server (self-managed) with administrative access
- A Linux host with Fluent Bit installed (can be the same host as Vault)
- Network connectivity between the Vault host and AWS

Log forwarding setup

HashiCorp Vault audit logs can be forwarded to Amazon S3 using the Vault file audit device combined with Fluent Bit. Vault supports three audit device types: **file**, **socket**, and **syslog**. We recommend the **file** audit device for highest reliability.

Instructions to setup Amazon S3 and Amazon SQS

Complete the following steps to configure the Amazon S3 and Amazon SQS infrastructure for HashiCorp Vault log ingestion.

Step 1: Create Amazon S3 bucket

Create an Amazon S3 bucket to store HashiCorp Vault logs. The bucket must reside in the same AWS Region where you plan to create the CloudWatch pipeline.

Step 2: Create Amazon SQS queue

Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue receives notifications when new log files are added to the bucket.

Step 3: Connect Amazon S3 to Amazon SQS

Configure the Amazon S3 bucket to send event notifications for `s3:ObjectCreated:*` events to the Amazon SQS queue.

Step 4: Configure Amazon SQS queue policy

Configure the Amazon SQS queue policy to allow the Amazon S3 bucket to send messages to the queue. Apply the following policy to your Amazon SQS queue:

```
{
  "Version": "2012-10-17",
  "Id": "AllowS3ToSQS",
```

```
  "Statement": [
    {
      "Sid": "AllowS3BucketNotification",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "<account-id>"
        }
      }
    }
  ]
}
```

Step 5: Configure Vault log export and Fluent Bit

Enable the Vault file audit device to write audit logs:

```
vault audit enable file file_path=/var/log/vault/audit.log
```

Configure JSON operational logs in the Vault server configuration file (`vault.hcl`):

```
log_level = "info"
log_format = "json"
```

Configure Fluent Bit to collect both audit and operational logs and deliver them to Amazon S3:

```
########################################
# SERVICE SECTION
########################################
[SERVICE]
    Flush        5
    Daemon       Off
    Log_Level    info
    storage.path /etc/fluent-bit/db/storage
    storage.sync normal
```

```
    storage.checksum off
    storage.backlog.mem_limit 10M

########################################
# INPUT SECTION
########################################
[INPUT]
    Name             tail
    Path             /var/log/vault/audit.log
    Tag              vault.audit
    Parser           json
    Read_from_Head   False
    DB               /etc/fluent-bit/db/vault-audit.db
    DB.Sync          Normal
    storage.type     filesystem
    Rotate_Wait      30
    Refresh_Interval 60

[INPUT]
    Name             tail
    Path             /var/log/vault/vault.log
    Tag              vault.operational
    Parser           json
    Read_from_Head   False
    DB               /etc/fluent-bit/db/vault-operational.db
    DB.Sync          Normal
    storage.type     filesystem
    Rotate_Wait      30
    Refresh_Interval 60

########################################
# OUTPUT SECTION
########################################
[OUTPUT]
    Name                s3
    Match               vault.*
    bucket              amzn-s3-demo-bucket
    region              <region>
    s3_key_format       /hashicorp-vault/%Y/%m/%d/%H/%M/%S.log.gz
    compression         gzip
    use_put_object      On
    store_dir           /etc/fluent-bit/db/s3
    total_file_size     1M
    log_key             log
```

```
    upload_timeout       60
```

Step 6: IAM permissions

Create an IAM policy with the following permissions for the Fluent Bit host to write objects to the Amazon S3 bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FluentBitS3Write",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/hashicorp-vault/*"
      ]
    }
  ]
}
```

Step 7: Verify

Verify the setup by confirming that log files appear in the Amazon S3 bucket and that Amazon SQS notifications are being generated. Check the Amazon SQS queue for messages indicating new object creation events.

Configuring the CloudWatch pipeline

- Choose HashiCorp Vault as the data source when creating the pipeline.
- Provide the Amazon SQS queue URL and IAM role ARN.
- Select the destination CloudWatch Logs log group.
- After you create the pipeline, data will be available in the selected log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. The following table shows the mapping between HashiCorp Vault log types and OCSF event classes. Events that are not mapped are forwarded as raw logs.

Event name	OCSF event class
Audit	API Activity [6003]
Operational	N/A (raw pass-through)

CloudWatch pipelines configuration for HashiCorp Vault

The HashiCorp Vault setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "hashicorp_vault"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. Must be "hashicorp_vault".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

Jamf Pro integration configuration

Jamf Pro is the flagship enterprise Mobile Device Management (MDM) and Enterprise Mobility Management (EMM) platform providing centralized lifecycle management for Apple devices across macOS, iOS, iPadOS, and tvOS. It provides structured per-device inventory records including hardware specifications, OS versions, security posture (FileVault, SIP, Gatekeeper), installed software, and user assignments. With CloudWatch Pipelines, you can use the Jamf Pro REST API to retrieve computer inventory data from your Jamf Pro tenant. The API exposes paginated REST endpoints with OAuth2 authentication that allow fetching device inventory records for monitoring and analysis.

Topics

- Source configuration for Jamf Pro
- CloudWatch pipelines configuration for Jamf Pro

Source configuration for Jamf Pro

Prerequisites

Before you begin, make sure you have the following:

- An active Jamf Pro tenant (Jamf Cloud or on-premises) with API access enabled.
- A Jamf Pro user account with API role permissions to access the Computer Inventory endpoint, or an API Client configured with the required OAuth2 scopes.
- An AWS account with permissions to create and manage CloudWatch Pipelines.
- An AWS account with required permissions to create, retrieve, and update secrets in AWS Secrets Manager.
- An AWS account with permissions to create and manage CloudWatch Logs log groups.

Integrating with Jamf Pro

To integrate Jamf Pro with CloudWatch Pipelines, complete the following high-level steps:

- Create an API Client (OAuth2) in the Jamf Pro Console with the required scopes.
- Note the Client ID and Client Secret.
- Identify your Jamf Pro instance URL.
- Store the credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with Jamf Pro as the data source.
- Verify data is flowing into the configured CloudWatch Logs log group.

Authenticating with Jamf Pro

To read the data, the pipeline needs to authenticate with your Jamf Pro tenant. Jamf Pro uses OAuth2 Client Credentials flow, which exchanges a Client ID and Client Secret for a short-lived bearer token (valid for 20 minutes) used for all subsequent API calls.

Configure Authentication for Jamf Pro

To configure authentication credentials for the pipeline:

1. Log in to the Jamf Pro Console and navigate to **Settings > System > API Roles and Clients**.
2. Choose **New API Role** and create a role with the following privileges:

- Read Computers
- Read Computer Inventory Collection

3. Choose **API Clients** > **New API Client**.
4. Assign the API Role created above to the client.
5. Enable the client and note the following credentials:
   - **Client ID**
   - **Client Secret**
6. Identify your **Jamf Pro instance URL** based on your deployment:
   - Jamf Cloud: `https://<instance_name>.jamfcloud.com`
   - On-premises: `https://<your-jamf-server-url>`
7. In AWS Secrets Manager, create a secret and store:
   - Client ID under the key `client_id`
   - Client Secret under the key `client_secret`

For more information, see Jamf Pro API Overview on the Jamf developer website.

Configuring the CloudWatch Pipeline

To configure the pipeline to read data, choose Jamf Pro as the data source. Provide the required information such as the Jamf Pro hostname (`hostname`), Client ID (`client_id`), and Client Secret (`client_secret`). After you create and activate the pipeline, computer inventory data from Jamf Pro will begin flowing into the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Jamf Pro events that map to Device Inventory Info (5001).

Device Inventory Info (5001)

Computer Inventory – Contains structured per-device records for managed Macs including hardware specifications, OS version, security posture (FileVault/SIP/Gatekeeper status), installed software, last contact time, and assigned user information.

> **Note**
>
> Events that do not match any OCSF event classes listed above – such as custom or newly introduced Jamf Pro event types that have not yet been mapped – are automatically passed through and sent directly to the configured sink in their original format without additional processing.

CloudWatch pipelines configuration for Jamf Pro

Collects computer inventory data from Jamf Pro using OAuth2 Client Credentials authentication.

Configure the Jamf Pro source with the following parameters:

```
source:
  jamf_pro:
    hostname: "<<your-instance>>.jamfcloud.com"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:jamf-pro-credentials:client_id}}"
        client_secret: "${{aws_secrets:jamf-pro-credentials:client_secret}}"
```

Parameters

`hostname` (required)

The Jamf Pro instance hostname for your tenant. For Jamf Cloud deployments, this follows the format `<instance_name>.jamfcloud.com`. For on-premises deployments, use your self-hosted Jamf Pro server hostname. Do not include `https://`.

`authentication.oauth2.client_id` (required)

The OAuth2 Client ID generated in the Jamf Pro Console. Stored in AWS Secrets Manager.

`authentication.oauth2.client_secret` (required)

The OAuth2 Client Secret generated in the Jamf Pro Console. Stored in AWS Secrets Manager.

> **Note**
>
> The `client_id` and `client_secret` values are retrieved from AWS Secrets Manager. These credentials can be generated in the Jamf Pro Console under Settings > System > API

Roles and Clients. The OAuth2 token is valid for 20 minutes and is automatically refreshed by the pipeline.

Jamf Protect integration configuration

Jamf Protect is an endpoint security solution purpose-built for macOS and iOS environments, providing real-time threat detection, behavioral analytics, and compliance monitoring across Apple device fleets. CloudWatch pipelines allow ingestion of Jamf Protect data into CloudWatch Logs.

Topics

- Source configuration for Jamf Protect
- CloudWatch pipelines configuration for Jamf Protect

Source configuration for Jamf Protect

Integrating with Jamf Protect

Jamf Protect integration uses Amazon S3 and Amazon SQS to ingest log data into CloudWatch pipelines. Jamf Protect supports native Amazon S3 delivery through Action Configurations (for macOS Security data) and Data Streams (for Jamf Security Cloud data), eliminating the need for an intermediate log forwarder such as Fluent Bit. Amazon SQS notifications alert the pipeline when new log objects arrive.

Prerequisites

- An AWS account with permissions to create Amazon S3 buckets, Amazon SQS queues, and IAM roles
- A Jamf Protect tenant with administrative access to configure Action Configurations or Data Streams
- IAM credentials for Jamf Protect to write to Amazon S3

Instructions to setup Amazon S3 and Amazon SQS

Complete the following steps to configure the Amazon S3 and Amazon SQS infrastructure for Jamf Protect log ingestion.

Step 1: Create Amazon S3 bucket

Create an Amazon S3 bucket to store Jamf Protect logs. The bucket must reside in the same AWS Region where you plan to create the CloudWatch pipeline.

Step 2: Create Amazon SQS queue

Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue receives notifications when new log files are added to the bucket.

Step 3: Connect Amazon S3 to Amazon SQS

Configure the Amazon S3 bucket to send event notifications for `s3:ObjectCreated:*` events to the Amazon SQS queue.

Step 4: Configure Amazon SQS queue policy

Configure the Amazon SQS queue policy to allow the Amazon S3 bucket to send messages to the queue. Apply the following policy to your Amazon SQS queue:

```
{
  "Version": "2012-10-17",
  "Id": "AllowS3ToSQS",
  "Statement": [
    {
      "Sid": "AllowS3BucketNotification",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:111122223333:queue-name",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

Step 5: Configure Jamf Protect to export logs to Amazon S3

Configure Jamf Protect to deliver log data directly to your Amazon S3 bucket.

macOS Security (Action Configuration)

1. In the Jamf Protect console, navigate to Action Configurations.
2. Create or edit an Action Configuration.
3. Configure the Amazon S3 destination with your bucket name, region, and IAM credentials.

Jamf Security Cloud (Data Streams)

1. In the Jamf Protect console, navigate to Integrations, then choose Data Streams.
2. Create a new Data Stream with Amazon S3 as the destination.
3. Provide the bucket name, region, and IAM credentials.

Step 6: Attach bucket policy

Apply the following bucket policy to your Amazon S3 bucket to allow the Jamf IAM user to write objects:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3Access",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/JAMF-IAM-USER"
      },
      "Action": [
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/jamf-protect-logs/*"
    },
    {
      "Sid": "S3ListBucket",
      "Effect": "Allow",
      "Principal": {
```

```
        "AWS": "arn:aws:iam::111122223333:user/JAMF-IAM-USER"
      },
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",
      "Condition": {
        "StringLike": {
          "s3:prefix": "jamf-protect-logs/*"
        }
      }
    }
  ]
}
```

Step 7: Verify

Verify the setup by confirming that log files appear in the Amazon S3 bucket and that Amazon SQS notifications are being generated. Check the Amazon SQS queue for messages indicating new object creation events.

Configuring the CloudWatch pipeline

- Choose Jamf Protect as the data source when creating the pipeline.
- Provide the Amazon SQS queue URL and IAM role ARN.
- Select the destination CloudWatch Logs log group.
- After you create the pipeline, data will be available in the selected log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. The following table shows the mapping between Jamf Protect data types and OCSF event classes.

Event name	OCSF event class
Alerts	Detection Finding [2004]
Telemetry	Pass-through
Unified Logs	Pass-through
Threat Events	Detection Finding [2004]

Event name	OCSF event class
Network Traffic	Network Activity [4001] / DNS Activity [4003]
Access Events (ZTNA)	Pass-through
Device Data	Device Inventory Info [5001]
App Insights	Software Inventory Info [5020]
Vulnerability Data	Vulnerability Finding [2002]

CloudWatch pipelines configuration for Jamf Protect

The Jamf Protect setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::111122223333:role/role-name"
    compression: "automatic"
    codec:
      ndjson:
    data_source_name: "jamf_protect"
    default_bucket_owner: "111122223333"
    bucket_owners:
      amzn-s3-demo-bucket: "111122223333"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.us-east-1.amazonaws.com/111122223333/queue-name"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. Must be "jamf_protect".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

JumpCloud integration configuration

JumpCloud is a cloud-based open directory platform that unifies identity, access, and device management across IT resources. It provides centralized directory services, Single Sign-On (SSO), Multi-Factor Authentication (MFA), RADIUS, LDAP, and device management for organizations managing hybrid environments. CloudWatch pipelines use version 1 of the JumpCloud Directory Insights API to retrieve security and audit events from your JumpCloud tenant. The pipeline requests events from every Directory Insights service, so all of the events that your organization records are delivered to the sink. Events from the directory, SSO, RADIUS, systems, LDAP, and alerts services are transformed into Open Cybersecurity Schema Framework (OCSF) format. Events from any other Directory Insights service are forwarded to the sink without transformation. The API provides a single POST endpoint with cursor-based pagination and API key authentication for comprehensive identity and access event data.

Topics

- Source configuration for JumpCloud
- CloudWatch pipelines configuration for JumpCloud

Source configuration for JumpCloud

Supported API and platform versions

Component	Version	Notes
Directory Insights API	v1	Security and audit event retrieval across all services
Open Cybersecurity Schema Framework (OCSF) schema	v1.5.0	Open Cybersecurity Schema Framework mapping
API key authentication	v1	API key sent in the `x-api-key` header

Prerequisites

Before you begin, make sure you have the following:

- An active JumpCloud organization with Directory Insights enabled. Directory Insights is included in some JumpCloud package plans. JumpCloud stores 90 days of event logs and removes logs that are older than 90 days.
- A JumpCloud administrator account with permissions to generate API keys.
- The JumpCloud API hostname for your organization's data residency region.
- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called “API caller permissions”.
- An AWS account with permissions to create and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called “Third-party sources (API Pull)”.
- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called “CloudWatch Logs API operations and required permissions for actions”.

Integrating with JumpCloud

To integrate CloudWatch pipelines with JumpCloud, complete the following high-level steps:

- Generate an API key in the JumpCloud Admin Console.

- Identify the API hostname for your organization's data residency region.
- Store the API key in AWS Secrets Manager.
- Create a destination CloudWatch Logs log group and the required source IAM role.
- Create a CloudWatch pipeline with JumpCloud as the data source.
- Verify that data is flowing into the configured CloudWatch Logs log group.

Authenticating with JumpCloud

JumpCloud uses API key authentication. The pipeline sends the API key as an `x-api-key` header with every request to the JumpCloud Directory Insights API. API keys are prefixed with `jca_` and are tied to an individual administrator account.

Configure authentication for JumpCloud

To configure authentication credentials for the pipeline:

1. Log in to the JumpCloud Admin Console with your administrator credentials.
2. Navigate to your administrator profile settings and select **My API Key**.
3. Generate a new API key. The key is prefixed with `jca_`.
4. Copy and securely store the API key immediately. You cannot view it again.
5. Identify the API hostname for your organization's data residency region:
   - US (default): `api.jumpcloud.com`
   - EU (European Union): `api.eu.jumpcloud.com`
   - IN (India): `api.in.jumpcloud.com`

   > **Note**
   >
   > The hostname must not include `https://`. Provide only the domain, such as `api.jumpcloud.com`.

6. In AWS Secrets Manager, choose **Store a new secret**, and then choose **Other type of secret**. Add the key-value pair `api_key` with the JumpCloud API key as the value. For example, name the secret `jumpcloud/pipeline-credentials`.

Configuring the CloudWatch pipeline

1. Create the destination CloudWatch Logs log group. For example, use `/aws/cloudwatch/pipelines/jumpcloud`.
2. Create a source IAM role with the required permissions. For information about the required policies and permissions, see the section called “CloudWatch pipelines IAM policies and permissions”.
3. Open the CloudWatch console, navigate to **Pipelines**, and choose JumpCloud as the data source.
4. Provide the API `hostname` for your organization's data residency region and the `api_key` stored in AWS Secrets Manager.
5. Select the destination log group.
6. Verify the destination log group resource policy. For a new log group, the console creates the resource policy automatically. If the log group already has a resource policy, the console warns you, and you must update that policy within five minutes of pipeline creation to include the required pipeline write permissions. For more information, see the section called “Resource policies”.
7. Wait a few minutes for the initial data pull, check the destination log group for incoming events, and monitor pipeline metrics for successful ingestion.

After you create and activate the pipeline, security and audit events from JumpCloud begin flowing into the selected CloudWatch Logs log group.

> **Note**
>
> When you create the pipeline using the AWS CLI or API, a resource policy is never created automatically, regardless of whether the log group is new or existing. You must create the CloudWatch Logs resource policy manually before the pipeline becomes active.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0. Each JumpCloud service maps to one or more of the following OCSF event classes.

> **Note**
>
> Events that do not match an OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

Authentication (3002)

- **Directory authentication events** (`directory`) – Admin portal login attempts, admin API key authentication, and user portal login attempts.
- **SSO authentication events** (`sso`) – User authentications to SSO applications. The authentication protocol of the target application is reported as SAML or OpenID.
- **Systems authentication events** (`systems`) – Device-level login attempts, including SSH and console login attempts, on JumpCloud-managed systems, and remote session start, join, and end events.
- **RADIUS authentication events** (`radius`) – RADIUS-based network authentication attempts with PAP, CHAP, and EAP support.
- **LDAP Bind authentication events** (`ldap`) – LDAP Bind operations representing directory authentication attempts.

Account Change (3001)

Directory account lifecycle events (`directory`) – User and administrator account lifecycle operations including creation, modification, deletion, activation, suspension, lockout, unlock, password change or reset, and MFA factor enable or disable operations.

Entity Management (3004)

- **Directory entity management events** (`directory`) – Create, read, update, and delete operations on directory objects including groups, policies, applications, SSO integrations, resource associations, SCIM provisioning, and platform configurations.
- **Systems entity management events** (`systems`) – System and device lifecycle management including creation, update, deletion, operating system upgrades, rollbacks, and Google EMM operations.
- **Alerts rule configuration events** (`alerts`) – Alert rule lifecycle management including rule creation, modification, and deletion.

Detection Finding (2004)

Alert detection finding events (`alerts`) – Alert creation, update, and status change events generated by JumpCloud security rules for device compliance and policy violations.

Datastore Activity (6005)

LDAP Search events (`ldap`) – LDAP directory query operations including search filter, base DN, and result tracking.

Known platform limitations

Limitation	Details	Impact
Directory Insights entitlement	Directory Insights is included in only some JumpCloud package plans. JumpCloud stores 90 days of event logs and removes logs that are older than 90 days.	If your plan does not include Directory Insights, no events are available until your JumpCloud account manager enables it. You cannot backfill events that are older than 90 days.
API key scope	API keys are tied to an individual administrator account.	If that administrator account is deactivated, the API key becomes invalid.
Data residency	The API hostname must match the data residency region that is configured for your organization.	If you use a hostname for another region, authentication fails.
Event delivery delay	Events can take a few minutes to appear in the Directory Insights API.	Near-real-time monitoring might not capture the most recent events.
API rate limits	JumpCloud does not publish explicit rate limit numbers for the Directory Insights API.	The connector handles HTTP 429 responses automatically by backing off and retrying.

Troubleshooting

Error	Likely cause	Resolution or connector behavior
`401 Unauthorized`	The API key is invalid or expired.	Verify that the `api_key` stored in AWS Secrets Manager is correct and that the associated administrator account is active. Regenerate the key if necessary.
`403 Forbidden`	Insufficient permissions, or the hostname does not match your region.	Make sure that the API key belongs to an administrator who has Directory Insights access. Verify that `hostname` matches your organization's data residency region.
`429 Too Many Requests`	The API rate limit was exceeded.	The connector automatically backs off and retries.
No data appears in the log group.	The pipeline resource policy was not added within five minutes of pipeline creation, before the pipeline became active.	Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.
Empty responses	Directory Insights is not enabled for your organization, or no events exist in the requested time range.	Verify that Directory Insights is enabled for your JumpCloud plan, and check that events exist within the configured `range` period.

No events from a specific service

- Verify that the service, such as SSO, RADIUS, or LDAP, is actively used in your JumpCloud organization. Services without activity do not generate events.
- Check that the service is enabled and configured in the JumpCloud Admin Console.

The API key becomes invalid unexpectedly

- API keys are tied to an individual administrator account. If that account is deactivated or deleted, or if its permissions change, the key becomes invalid.
- Regenerate the API key from an active administrator account, and then update the secret in AWS Secrets Manager.

Events are missing after a data residency change

- If your organization changed data residency regions, update the `hostname` parameter to the API hostname for the new region.

CloudWatch pipelines configuration for JumpCloud

The following pipeline configuration collects security and audit events from JumpCloud using API key authentication, and transforms the events from the directory, Single Sign-On (SSO), RADIUS, systems, LDAP, and alerts services into Open Cybersecurity Schema Framework (OCSF) format. Replace the placeholder values shown between << and >> with values for your environment.

```
extension:
  aws:
    secrets:
      jumpcloud_credentials:
        secret_id: <<arn:aws:secretsmanager:<region>:<account-id>:secret:<secret-
name>>>
        region: <<region>>
        sts_role_arn: <<arn:aws:iam::<account-id>:role/<role-name>>>
pipeline:
  source:
    jumpcloud:
      hostname: "api.jumpcloud.com"
      authentication:
        api_key: "${{aws_secrets:jumpcloud_credentials:api_key}}"
      range: "P90D"
  sink:
    - cloudwatch_logs:
        log_group: <<jumpcloud-log-group>>
  processor:
    - ocsf:
        schema:
```

```
          jumpcloud: null
        version: "1.5"
        mapping_version: 1.5.0
```

The AWS extension configures AWS Secrets Manager access for securely retrieving credentials at runtime. Define a named secret block, such as `jumpcloud_credentials`, with `secret_id` set to the ARN of your AWS Secrets Manager secret, `region` set to the AWS Region where the secret is stored, and `sts_role_arn` set to the IAM role ARN to assume when accessing the secret. Reference credentials in the pipeline using `${{aws_secrets:<name>:<key>}}`. The name in the `aws_secrets` reference must match the secret block name under `extension.aws.secrets`. For the complete configuration reference, see the AWS Secrets Manager extension.

Source parameters

`hostname` (required)

The JumpCloud API hostname for your organization's data residency region. Use `api.jumpcloud.com` for the US, `api.eu.jumpcloud.com` for the European Union, or `api.in.jumpcloud.com` for India. Do not include `https://`.

`authentication.api_key` (required)

The JumpCloud API key, prefixed with `jca_`, for authenticating API requests. Store the key in AWS Secrets Manager and reference it using `${{aws_secrets:jumpcloud_credentials:api_key}}`.

`range` (optional)

The lookback duration for events on the first run, in ISO 8601 duration format. The minimum value is `PT1H`, the maximum value is `P90D`, and the default is `P90D`. JumpCloud retains 90 days of events, so you cannot backfill events that are older than 90 days.

Note

The `api_key` value is retrieved from AWS Secrets Manager. Generate the API key in the JumpCloud Admin Console under your administrator profile settings by choosing **My API Key**.

Sink parameters

`log_group` (required)

The name of the destination CloudWatch Logs log group, such as `jumpcloud-log-group`. For complete configuration options, see the CloudWatch Logs sink.

Processor parameters

`schema.jumpcloud` (required)

Set this parameter to `null`. The presence of the key activates the JumpCloud schema mapping and transforms ingested events into OCSF format.

`version` (required)

The OCSF schema version. Set this parameter to `"1.5"`.

`mapping_version` (required)

The mapping version. Set this parameter to `1.5.0`. For the full parameter reference, see the OCSF processor.

Microsoft 365 integration configuration

This section describes how to configure Microsoft 365 as a data source for CloudWatch Pipeline.

Topics

- Source configuration for Microsoft 365
- CloudWatch pipelines configuration for Microsoft 365

Source configuration for Microsoft 365

Integrating with Microsoft 365

Microsoft 365 is a product family of productivity software, collaboration, and cloud-based services owned by Microsoft. CloudWatch Pipeline uses the Office 365 Management Activity API to retrieve information about user, admin, system, and policy actions and events from Office 365 and Microsoft Entra activity logs. Office 365 Management Activity API (also known as the Unified Auditing API) is a part of Office 365 security and compliance offerings. Customers and partners

can use this information to create new or enhance existing operations, security, and compliance-monitoring solutions for the enterprise.

Authenticating with Office 365 Management Activity API

To retrieve the activities Office 365, pipelines needs to authenticate with your account. Follow the instructions in Office 365 Management APIs:

- Register an application in Azure with Supported account types, Accounts in this organizational directory only (Single tenant). After registration is complete, note down the Application (client) ID and Directory (tenant) ID.
- Generate a new key for your application. Key is also known as client secret, which are used when exchanging an authorization code for an access token.
- In the AWS Secrets Manager, create a secret and store the Application (client) ID under the key `client_id` and the client secret under the key `client_secret`
- Specify the permissions your application requires to access the Office 365 Management APIs. The permissions you need are:
  - ActivityFeed.Read: Required for all the audit content types you listed, including Audit.AzureActiveDirectory, Audit.Exchange, Audit.SharePoint, and Audit.General.
  - ActivityFeed.ReadDlp: Required specifically for the DLP.All content type
- Before you can access data through the Office 365 Management Activity API, you must enable unified audit logging for your Office 365 organization. You do this by turning on the Office 365 audit log. For instructions, see Turn Office 365 audit log search on or off.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read activities from Office 365, choose Microsoft 365 as the data source. Fill in the required information like Tenant Id using Directory (tenant) ID and the secret where `client_id` and `client_secret` are stored. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. Actions from various workloads like Exchange, SharePoint, Teams, and Azure Active Directory are mapped to Account Change (3001), Authentication (3002), User Access Management (3005), Group Management (3006), Email Activity (4009), Web Resources Activity (6001), File Hosting Activity (6006), Application Lifecycle

(6002), (2003), Detection Finding (2004), Incident Finding (2005), Vulnerability Finding (2002) and Unknown (0).

Compliance Finding

Compliance Finding contains the following actions:

- ApplyRecordLabel
- ComplianceSettingChanged
- ExclusionConfigurationDeleted
- NewComplianceTag
- NewRetentionCompliancePolicy
- NewRetentionComplianceRule
- CreateRulePackage
- CreateSuppressionRule
- ApproveDisposal
- RemoveComplianceTag
- SetComplianceTag
- SetRestrictiveRetentionUI
- SupervisionPolicyCreated
- SupervisionPolicyUpdated
- SupervisionPolicyDeleted
- HoldUpdated
- HoldCreated
- HoldRemoved
- DlpInfo

Detection Finding

Detection Finding contains the following actions:

- FileMalwareDetected
- DocumentSensitivityMismatchDetected

- TIMailData
- DeviceOffBoarding
- AddIndicator
- ChangeCustomDetectionRuleStatus
- CreateCustomDetection
- DeleteIndicator
- EditIndicator
- MonitoringAlertUpdated
- RunCustomDetection
- Trigger CMD Agent Canary check.
- DlpRuleMatch
- AlertEntityGenerated
- AlertTriggered

Incident Finding

Incident Finding contains the following actions:

- AddCommentToIncident
- AddTagsToIncident
- AssignUserToIncident
- CollectInvestigationPackage
- EditIncidentClassification
- RemediationActionAdded
- RemediationActionUpdated
- RemoveTagsFromIncident
- UnAssignUserFromIncident
- UpdateIncidentStatus
- CaseUpdated
- CaseAdded
- CaseRemoved

Account Change

Account Change contains the following actions:

- Add member to role
- Add service principal
- Add user
- Added role
- Change user license
- Change user password
- Delete user
- Deleted app permission
- Deleted role
- Edited global role assignment
- Edited role
- NetworkUserSuspended
- Remove member from role
- Remove delegation entry
- Reset user password
- Set force change user password
- AdministratorAddedToTermStore
- AdministratorDeletedFromTermStore
- AlertNotificationsRecipientDeleted
- CaseAdminUpdated
- CaseAdminAdded
- CaseAdminRemoved
- User added

Authentication

Authentication contains the following actions:

- MailboxLogin

- ClockedIn
- ClockedOut
- TeamsSessionStarted
- Logon
- SignInEvent
- SSOUserCredentialsSet
- User logged in
- UserLoggedIn
- UserLoggedOff
- UserLoginFailed

User Access Management

User Access Management contains the following actions:

- Add-MailboxPermission
- ModifyFolderPermissions
- Remove-MailboxPermission
- ApplicableAdaptiveScopeChange
- CaseMemberAdded

Group Management

Group Management contains the following actions:

- RemovedFromSecureLink
- BotAddedToTeam
- BotRemovedFromTeam
- MemberAdded
- MemberRemoved
- MemberRoleChanged
- ScheduleGroupAdded
- ScheduleGroupEdited

- ScheduleGroupDeleted
- TeamCreated
- TeamDeleted
- Add group
- Add member to group
- Created group
- Delete group
- Deleted group
- Edited group memberships
- Edited group
- GroupCreation
- GroupDeletion
- GroupRemoved
- GroupAdded
- GroupUpdated
- RemovedFromGroup
- AddedToGroup
- Remove member from group
- RemoveSpecificResponder
- RosterMemberAdded
- RosterMemberDeleted
- CaseMemberUpdated
- CaseMemberRemoved
- Team added
- Team deleted
- UserAddedToGroup
- UserRemovedFromGroup

Email Activity

Email Activity contains the following actions:

- Send
- SendAs
- SendOnBehalf
- MessageDeletedNotification
- QuarantineDelete
- QuarantineExport
- QuarantinePreview
- QuarantineRelease
- QuarantineReleaseRequest
- QuarantineReleaseRequestDeny
- QuarantineViewHeader
- SupervisionRuleMatch
- SupervisoryReviewTag
- SupervisoryReviewOLAudit

Web Resources Activity

Web Resources Activity contains the following actions:

- UpdateCalendarDelegation
- AddFolderPermissions
- Copy
- Create
- New-InboxRule
- SoftDelete
- Move
- MailItemsAccessed
- MoveToDeletedItems
- Set-InboxRule
- HardDelete
- UpdateInboxRules

- Update
- LockRecord
- UnlockRecord
- SearchQueryPerformed
- PageViewed
- PageViewedExtended
- FolderCreated
- ClientViewSignaled
- PagePrefetched
- FolderModified
- ListColumnCreated
- ListContentTypeCreated
- ListItemCreated
- Site ContentType Created
- List Column Deleted
- ListCreated
- List Item Deleted
- SiteColumnDeleted
- ListDeleted
- ListContentTypeDeleted
- ListRestored
- SiteColumnCreated
- ListItemRecycled
- ListItemDeleted
- ListItemRestored
- ListContentTypeUpdated
- ListUpdated
- ListViewed
- SiteContentTypeDeleted
- ListItemUpdated

- SiteColumnUpdated
- AccessRequestAccepted
- ListColumnUpdated
- SiteContentTypeUpdated
- AccessRequestCreated
- PermissionLevelAdded
- CompanyLinkCreated
- AnonymousLinkCreated
- SharingInvitationAccepted
- SecureLinkCreated
- SharingInvitationCreated
- SecureLinkDeleted
- CompanyLinkRemoved
- AccessRequestDenied
- AnonymousLinkRemoved
- AccessRequestUpdated
- SharingSet
- AnonymousLinkUpdated
- SharingInvitationBlocked
- AnonymousLinkUsed
- SecureLinkUsed
- CompanyLinkUsed
- SharingRevoked
- AddedToSecureLink
- SharingInvitationUpdated
- SharingInvitationRevoked
- ExemptUserAgentSet
- AllowedDataLocationAdded
- SiteGeoMoveCancelled
- AllowGroupCreationSet

- CustomizeExemptUsers
- DeviceAccessPolicyChanged
- NetworkAccessPolicyChanged
- SiteCollectionCreated
- SiteDeleted
- SendToConnectionRemoved
- SiteGeoMoveCompleted
- SharingPolicyChanged
- PreviewModeEnabledSet
- HubSiteOrphanHubDeleted
- SendToConnectionAdded
- HubSiteJoined
- SiteCollectionQuotaModified
- LegacyWorkflowEnabledSet
- OfficeOnDemandSet
- NewsFeedEnabledSet
- PeopleResultsScopeSet
- AllowedDataLocationDeleted
- SiteRenamed
- HubSiteRegistered
- HostSiteSet
- GeoQuotaAllocated
- HubSiteUnjoined
- HubSiteUnregistered
- SiteCollectionAdminAdded
- PermissionLevelsInheritanceBroken
- SharingInheritanceBroken
- SiteGeoMoveScheduled
- WebRequestAccessModified
- WebMembersCanShareModified

- PermissionLevelModified
- PermissionLevelRemoved
- SitePermissionsModified
- SiteCollectionAdminRemoved
- SiteAdminChangeRequest
- SharingInheritanceReset
- BreakEnded
- ChannelAdded
- BreakStarted
- ChannelDeleted
- ChannelOwnerResponded
- ChatRetrieved
- ChannelSettingChanged
- ChatCreated
- ChatUpdated
- ConnectorAdded
- ConnectorRemoved
- ConnectorUpdated
- CreateUpdateRequest
- EditUpdateRequest
- FailedValidation
- InviteeResponded
- InviteSent
- MeetingDetail
- MeetingParticipantDetail
- MessageCreatedHasLink
- MessageDeleted
- MessageCreatedNotification
- MessageEditedHasLink
- MessageHostedContentRead

- MessageRead
- MessageReadReceiptReceived
- MessageHostedContentsListed
- MessageSent
- MessagesExported
- MessageUpdated
- MessageUpdatedNotification
- OffShiftDialogAccepted
- MessagesListed
- OpenShiftAdded
- OpenShiftDeleted
- OpenShiftEdited
- PerformedCardAction
- RequestAdded
- RequestRespondedTo
- RequestCancelled
- ScheduleSettingChanged
- ScheduleShared
- SensitivityLabelApplied
- ScheduleWithdrawn
- SensitivityLabelChanged
- SensitivityLabelRemoved
- SharingRestored
- ShiftAdded
- ShiftDeleted
- ShiftEdited
- SubscribedToMessages
- TabAdded
- SubmitUpdate
- TabRemoved

- TabUpdated
- TeamSettingChanged
- TeamsTenantSettingChanged
- TerminatedSharing
- TimeClockEntryDeleted
- TimeClockEntryAdded
- TimeClockEntryEdited
- TimeOffAdded
- TimeOffEdited
- ViewUpdate
- TimeOffDeleted
- TranscriptsExported
- AccessedOdataLink
- AcceptedSharingLinkOnFolder
- Add delegation entry.
- Add domain to company.
- Add service principal credentials.
- Add partner to company.
- Update service principal.
- AddedDataLossPreventionEvaluationResult
- AddFormCoauthor
- AddReviewer
- AddSpecificResponder
- Admin allowed third party apps
- Admin modified app owner
- Admin modified app permissions
- Admin set app as featured
- Admin set bypass consent state
- Admin set conditional access
- Admin set desired logical name

- Admin set quarantine state
- AlertExcelDownloaded
- AlertNotificationsRecipientAdded
- AllowAnonymousResponse
- AllowShareFormForCopy
- AppBypassInformationBarrier
- CanceledQuery
- Check PowerShell Execution Policy
- ClassificationDefinitionDeleted
- ClassificationAdded
- ClassificationDefinitionUpdated
- ClassificationDeleted
- ClassificationDefinitionCreated
- CollectionHardDeleted
- CollectionCreated
- CollectionRenamed
- CollectionSoftDeleted
- Commented on video
- CommunityAccessFailure
- CollectionUpdated
- Consented to the app's APIs
- ConnectToExcelWorkbook
- Create LogCollection Request
- Create new work items (Scheduler)
- ConsentModificationRequest
- Create Remote Action Operation in Acti...
- CreateComment
- CreateForm
- CreateResponse
- Dashboard created

- Dashboard deleted
- Dashboard updated
- Data exported
- DataAccessRequestOperation
- DataExport
- DataShareCreated
- DeleteAllResponses
- DeleteCustomDetection
- Deleted video
- DeletedResult
- DeleteSummaryLink
- DisableCollaboration
- DisableSpecificResponse
- DisallowShareFormForCopy
- DisableSuppressionRule
- DisallowAnonymousResponse
- EditCustomDetection
- Edited app
- Edited app permission
- Edited global role assignment
- Edited channel
- Edited tenant settings
- Edited group
- Edited user settings
- Edited role
- Edited video permission
- EditForm
- Edited video
- EditRulePackage
- EnableSameOrgCollaboration

- EditSuppressionRule
- EnableSpecificCollaboaration
- EnableSpecificResponse
- EnableSuppressionRule
- EnableWorkOrSchoolCollaboration
- EntityCreated
- EntityDeleted
- EntityRemediatorConfigurationUpdated
- EntityUpdated
- ExclusionConfigurationAdded
- ExclusionConfigurationUpdated
- ExecutedQuery
- ExportForm
- ExtendRetention
- FileUpdateDescription
- FileUpdateName
- FileVisited
- FolderSharingLinkShared
- SharingLinkUsed
- SharingLinkCreated
- GenerateCopyOfLakeData
- Get text track
- Get transcript
- Get video
- GetSummaryLink
- GlossaryTermAssigned
- GlossaryTermCreated
- GlossaryTermDisassociated
- GlossaryTermDeleted
- GlossaryTermUpdated

- Goals policy updated
- Group view
- InformationBarriersInsightsReportOneDr...
- InformationBarriersInsightsReportSched...
- InformationBarriersInsightsReportShare...
- InformationBarriersInsightsReportCompl...
- Liked video
- Linked on Video
- LinkedEntityCreated
- LinkedEntityDeleted
- LinkedEntityUpdated
- ListForms
- Marked app as Featured
- Marked app as Hero
- MarkedMessageChanged
- ReactedToMessage
- MeetingExclusionCreated
- MessageCreated
- MessageAccessFailure
- MessageViewed
- MonitoringAlertNotificationRecipientAd...
- MonitoringAlertNotificationRecipientDe...
- MovedFormIntoCollection
- MovedFormOutofCollection
- NetworkConfigurationUpdated
- NetworkSecurityConfigurationUpdated
- MoveForm
- NewAdaptiveScope
- NotificationConfigurationUpdated
- OCE Run Commands on VM

- OKR or Project created
- OKR or Project deleted
- OKR or Project updated
- Organization created
- Organization integrations updated
- Organization settings updated
- PlanCreated
- PlanCopied
- PlanDeleted
- PlanRead
- Post Remote Action Operation
- PlanListRead
- PreviewForm
- PlanModified
- ProcessProfileFields
- ProjectCreated
- ProjectAccessed
- ProInvitation
- ProjectDeleted
- ProjectForTheWebRoadmaptSettings
- ProjectForTheWebProjectSettings
- ProjectListAccessed
- ProjectUpdated
- RelabelItem
- ReleaseFromIsolation
- Remove domain from company.
- Remove partner from company.
- Remove service principal credentials.
- RemoveAdaptiveScope
- RemoveAppRestrictions

- RemoveFormCoauthor
- RemoveRetentionComplianceRule
- RemoveRetentionCompliancePolicy
- ReporterConfigurationUpdated
- RestrictAppExecution
- RoadmapAccessed
- RoadmapCreated
- RoadmapDeleted
- RoadmapItemAccessed
- RoadmapItemCreated
- RoadmapItemDeleted
- RoadmapItemUpdated
- RoadmapUpdated
- RosterCreated
- RosterDeleted
- RosterSensitivityLabelUpdated
- Run hybrid AADJ extension
- RunLiveResponseApi
- SensorCreated
- SensorConfigurationUpdated
- SensorDeleted
- SensorDeploymentAccessKeyUpdated
- SensorDeploymentAccessKeyReceived
- Set company contact information
- Set channel thumbnail
- Set delegation entry
- Set company information
- Set domain authentication
- Set federation settings on domain
- Set DirSyncEnabled flag

- Set license properties
- Set password policy
- SetAdaptiveScope
- SetAdvancedFeatures
- SetRetentionCompliancePolicy
- SiteIBModeChanged
- Shared video
- SiteIBModeSet
- SetRetentionComplianceRule
- SiteIBSegmentsChanged
- SiteIBSegmentsRemoved
- SiteIBSegmentsSet
- SiteSensitivityLabelApplied
- SensitivityLabelUpdated
- SiteSensitivityLabelChanged
- SiteSensitivityLabelRemoved
- SoftDeleteSettingsUpdated
- SPOIBIsDisabled
- SPOIBIsEnabled
- SubmitResponse
- SubTaskCreated
- SubTaskDeleted
- SubTaskUpdated
- SupervisorAdminToggled
- SyslogServiceConfigurationUpdated
- TaggingConfigurationUpdated
- TaskAccessed
- TaskAssigned
- TaskCompleted
- TaskDeleted

- TaskCreated
- TaskListCreated
- TaskListRead
- TaskListUpdated
- TaskModified
- TaskRead
- TaskUpdated
- Team updated
- TenantSettingsUpdated
- Trigger device remediation
- Trigger generic action by SaaF
- Trigger generic action
- Trigger generic action with options
- Unliked video
- Trigger orchestrator
- Update group.
- Update user.
- UpdatedDataAccessSetting
- UpdatedOrganizationBriefingSettings
- UpdatedOrganizationMyAnalyticsSettings
- Update domain.
- UpdatedPrivacySetting
- UpdatedUserBriefingSettings
- UpdatedUserMyAnalyticsSettings
- UpdateFormSetting
- UpdatePhishingStatus
- UpdateResponse
- UpdateUsageReportsPrivacySetting
- UpdateUserSetting
- URbacAuthorizationStatusChanged

- UserInvited
- UserSuspension
- Viewed video
- Verify domain
- Verify email verified domain
- ViewedExplore
- ViewForm
- ViewResponses
- ViewRuntimeForm
- ViewResponse
- VpnConfigurationUpdated
- WorkspaceCreated
- WorkspaceDeleted
- WorkspaceAlertThresholdLevelUpdated
- SearchUpdated
- SearchPermissionUpdated
- PreviewItemListed
- SearchCreated
- SearchPermissionCreated
- SearchRemoved
- SearchExportDownloaded
- SearchPreviewed
- SearchPermissionRemoved
- SearchResultsPurged
- RemovedSearchResultsSentToZoom
- RemovedSearchPreviewed
- RemovedSearchExported
- RemovedSearchResultsPurged
- SearchResultsSentToZoom
- SearchReportRemoved

- SearchStarted
- SearchReport
- ThreadViewed
- CaseViewed
- SearchViewed
- ViewedSearchExported
- SearchStopped
- ViewedSearchPreviewed
- AddWorkingSetQueryToWorkingSet
- AddQueryToWorkingSet
- AddNonOffice365DataToWorkingSet
- AnnotateDocument
- LoadComparisonJob
- RunAlgo
- CreateWorkingSet
- CreateWorkingSetSearch
- CreateTag
- DeleteTag
- UpdateTag
- DeleteWorkingSetSearch
- UpdateCaseSettings
- UpdateWorkingSetSearch
- PreviewWorkingSetSearch
- TagJob
- LabelContentExplorerAccessedItem
- AccessInvitationAccepted
- AccessInvitationCreated
- AccessInvitationExpired
- AccessInvitationRevoked
- AccessInvitationUpdated

- AccessRequestApproved
- AccessRequestRejected
- AppCatalogCreated
- AuditPolicyUpdate
- ActivationEnabled
- AuditPolicyRemoved
- AzureStreamingEnabledSet
- CollaborationTypeModified
- CreateSSOApplication
- ConnectedSiteSettingModified
- CustomFieldOrLookupTableCreated
- CustomFieldOrLookupTableDeleted
- CustomFieldOrLookupTableModified
- DelegateModified
- DelegateRemoved
- DefaultLanguageChangedInTermStore*
- eDiscoveryHoldApplied
- eDiscoveryHoldRemoved
- eDiscoverySearchPerformed
- EngagementAccepted
- EngagementModified
- EnterpriseCalendarModified
- EngagementRejected
- EntityForceCheckedIn
- LanguageAddedToTermStore
- LookAndFeelModified
- LanguageRemovedFromTermStore
- MaxQuotaModified
- MaxResourceUsageModified
- MySitePublicEnabledSet

- ODBNextUXSettings
- PermissionSyncSettingModified
- PermissionTemplateModified
- PortfolioDataAccessed
- PortfolioDataModified
- ProjectCheckedOut
- ProjectCheckedIn
- ProjectModified
- ProjectPublished
- ProjectWorkflowRestarted
- PWASettingsAccessed
- ProjectForceCheckedIn
- PWASettingsModified
- QueueJobStateModified
- QuotaWarningEnabledModified
- RenderingEnabled
- ReportingAccessed
- ResourceCheckedIn
- ResourceAccessed
- ReportingSettingModified
- ResourceCreated
- ResourceCheckedOut
- ResourceModified
- ResourcePlanCheckedInOrOut
- ResourceDeleted
- ResourcePlanModified
- ResourcePlanPublished
- ResourceForceCheckedIn
- ResourceWarningEnabledModified
- ResourceRedacted

- SSOGroupCredentialsSet
- SearchCenterUrlSet
- SecondaryMySiteOwnerSet
- SecurityCategoryModified
- SecurityGroupModified
- SiteCollectionAdminAdded*
- StatusReportModified
- SyntexBillingSubscriptionSettingsChang...
- TaskStatusAccessed
- TaskStatusApproved
- TaskStatusRejected
- TaskStatusSubmitted
- TaskStatusSaved
- TimesheetRejected
- TimesheetApproved
- TimesheetSaved
- TimesheetSubmitted
- TimesheetAccessed
- UpdateSSOApplication
- WorkflowModified
- DlpRuleUndo
- AlertUpdated
- SensitivityLabelPolicyMatched
- CopilotInteraction
- Channel view
- Deleted video comment
- Deleted channel
- Created channel
- Created video
- User deactivated

- User deleted

Application Lifecycle

Application Lifecycle contains the following actions:

- AppDeletedFromCatalog
- AppPublishedToCatalog
- AppInstalled
- AppUninstalled
- AppUpdatedInCatalog
- AppUpgraded
- DeletedAllOrganizationApps
- WorkforceIntegrationAdded
- AddDevicesToBackfill Operation
- AddDevicesToReinstall Operation
- Admin deleted app
- Admin restored deleted app
- Create VmExtention Request
- Created app
- Deleted app
- Deleted app version
- Execute AppHealthPlugin
- Install RD agent
- Update device.
- MigrationJobCompleted
- Patched app
- Published app
- Remove service principal.
- Removed app as Featured
- Removed app as Hero

- TriggerClientAgentCheckBulkAction Opera...
- Launched app
- LaunchPowerApp
- DeleteSSOApplication

File Hosting Activity

File Hosting Activity contains the following actions:

- UpdateFolderPermissions
- FileCheckedIn
- FileCheckedOut
- FileCopied
- FileAccessedExtended
- FileDeletedSecondStageRecycleBin
- FileDeleted
- FileAccessed
- FileDeletedFirstStageRecycleBin
- RecordDelete
- FileDownloaded
- FileCheckOutDiscarded
- FileModified
- FileModifiedExtended
- FilePreviewed
- FileRecycled
- FolderRecycled
- FileVersionsAllMinorsRecycled
- FileMoved
- FileVersionRecycled
- FileUploaded
- FileRenamed

- FileVersionsAllRecycled
- FileRestored
- FolderDeleted
- FolderDeletedFirstStageRecycleBin
- FolderMoved
- FolderCopied
- FolderDeletedSecondStageRecycleBin
- FolderRenamed
- FolderRestored
- RecordingExported
- ManagedSyncClientAllowed
- FileSyncDownloadedFull
- FileSyncDownloadedPartial
- FileSyncUploadedFull
- UnmanagedSyncClientBlocked
- FileSyncUploadedPartial
- AttachmentDeleted
- AttachmentUpdated
- AttachmentCreated
- DataShareDeleted
- Deleted text track
- Deleted thumbnail
- DomainControllerCoverageExcelDownloaded
- DownloadCopyOfLakeData
- Downloaded video
- DownloadedReport
- DownloadOffboardingPkg
- DownloadFile
- DownloadOnboardingPkg
- FileAccessFailure

- FileCreated
- FileSensitivityLabelChanged
- FileSensitivityLabelApplied
- FileSensitivityLabelRemoved
- FileShared
- WACTokenShared
- LiveResponseGetFile
- LogsCollection
- AddRemediatedData
- BurnJob
- DownloadDocument
- ExportJob
- ErrorRemediationJob
- TagFiles
- PreviewItemRendered
- ViewDocument
- FileFetched
- FileViewed
- SharedLinkCreated
- SharedLinkDisabled
- SharingInvitationAccepted*
- SyncGetChanges
- Restored app version
- RunAntiVirusScan
- StopAndQuarantineFile
- Uploaded text track
- Upload folder to blob
- Uploaded thumbnail
- Uploaded video
- UploadedOrgData

- ReportDownloaded
- PreviewItemDownloaded
- SearchExported
- Published solution canvas app version

CloudWatch pipelines configuration for Microsoft 365

Collects log data from Microsoft Office 365 Management API using OAuth2 authentication.

Configure the Microsoft Office 365 source with the following parameters:

```
source:
  microsoft_office365:
    tenant_id: "<example-tenant-ID>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`tenant_id` (required)

The Microsoft Entra ID tenant ID for your Office 365 organization.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Microsoft Office 365 Management API authentication. Can reference AWS Secrets Manager using `${{aws_secrets:secret-name:key}}` syntax.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Microsoft Office 365 Management API authentication. Can reference AWS Secrets Manager using `${{aws_secrets:secret-name:key}}` syntax.

> **Note**
>
> Store sensitive credentials like client IDs and secrets in AWS Secrets Manager and reference them using the `${{aws_secrets:secret-name:key}}` syntax in your configuration.

Netskope integration configuration

Netskope is a cloud-native Security Service Edge (SSE) and SASE platform that provides real-time data and threat protection for cloud services, websites, and private applications. CloudWatch pipelines use the Netskope REST API v2 endpoints to retrieve security events and alerts from your Netskope tenant, including application activity, network traffic, audit events, infrastructure logs, and threat detections across DLP, malware, and policy violations.

Topics

- Source configuration for Netskope
- CloudWatch pipelines configuration for Netskope

Source configuration for Netskope

Integrating with Netskope

Netskope is a cloud-native Security Service Edge (SSE) and SASE platform that provides real-time data and threat protection for cloud services, websites, and private applications. CloudWatch Pipeline uses the Netskope REST API v2 endpoints to retrieve security events and alerts from your Netskope tenant. The REST API v2 provides access to security event and alert logs through dedicated endpoints for each log type: application, page, network, infrastructure, audit, incident, endpoint, and alert. Alert logs cover threat detections and policy violations across alert types such as DLP, malware, malsite, policy, compromised credential, and UBA.

Authenticating with Netskope

To read Netskope event and alert logs, the pipeline needs to authenticate with your tenant using a REST API v2 token issued through a Service Account under the RBACv3 framework. Follow these steps to create a Service Account and generate an API token:

1. Log in to your Netskope Admin Console at https://<your-tenant>.goskope.com.
2. Navigate to **Settings** > **Administration** > **Administrators & Roles**.
3. Choose the **Roles** tab, then choose **Create Role**.
4. Enter a Role Name (for example, "CloudWatch-API-Role") and configure the following functional area permissions:
   - **Steering**: Application Events, Page Events, Network Events, Infrastructure Events, Incident Events, Endpoint Events, Alerts — all set to View.

- **Administration**: Audit Log — set to View.
- **Access Control**: Infrastructure — set to View.
- **DLP**: DLP Incident — set to View.

5. Choose **Save** to create the role.
6. Choose the **Administrators** tab, then choose the **Service Accounts** button.
7. Choose **New Service Account** and configure:
   - **Service Account Name**: Enter a descriptive name (for example, "CloudWatch-Collector").
   - **Role**: Select the role created in step 5 (for example, "CloudWatch-API-Role").
   - **Expire In**: Set an appropriate expiration period (for example, 365 days).
8. Choose **Create**. A dialog displays the generated REST API token. Copy this token immediately — it will not be shown again.
9. In AWS Secrets Manager, create a secret and store the API token.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read event and alert logs from Netskope, choose Netskope as the data source. Fill in the required information such as your tenant hostname and the AWS Secrets Manager secret ARN for your credentials where api_token is stored. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following events that map to Authentication (3002), Entity Management (3004), Account Change (3001), Network Activity (4001), Detection Finding (2004), Data Security Finding (2006), File Hosting Activity (6006), and Device Inventory Info (5001). Events that are not listed but pulled are not mapped to OCSF and will be forwarded to the sink as raw logs.

Authentication (3002) contains the following events from two event types:

Application Events, using the "activity" field. The supported activity values are:

- Login Attempt
- Login Failed
- Login Successful
- Logout

- Other Events which contain the keyword "login"

Audit Events, using the "audit_log_event" field. The supported audit_log_event values are:

- Login Failed
- Login Successful
- Logout Successful
- SSO Login Failed
- SSO Login Successful
- SSO Login Successful by Netskope Support
- SSO Login Failed by Netskope Support
- Admin logged out because of successive login failures

Entity Management (3004) contains the following events from Audit Events:

- Created new inline policy
- Set dedicated egress ip policy
- Created new rbi template
- Created new tunnel group
- Created new introspection policy
- Next Gen CASB API instance created
- Created a new Next Gen CASB API policy
- Next Gen CASB API retroscan created
- Edited inline policy
- Update default actions for inline policies
- Edited rbi template
- Edited tunnel group
- Edit introspection policy record
- Next Gen CASB API instance updated
- Edited Next Gen CASB API policy
- Next Gen CASB API retroscan edited

- Deleted inline policy
- Deleted rbi template
- Deleted tunnel group
- Deleted introspection policy
- Next Gen CASB API instance deleted
- Deleted Next Gen CASB API policy
- Next Gen CASB API retroscan deleted
- Pushed inline policy
- Pushed rbi template
- Pushed tunnel groups
- Applied Phoenix policy record(s)
- Pushed Introspection policies
- Pushed Next Gen CASB API policies
- Next Gen CASB API retroscan stopped
- Next Gen CASB API retroscan paused

Account Change (3001) contains the following events from Audit Events:

- Created new admin
- Added SSO Admin
- Created new support admin
- Enabled admin
- Password Change Failed Attempt
- Password Change Successful
- Reset password
- Disabled admin
- Deleted admin
- Deleted Netskope SSO admin
- Enabled Netskope Support SSO
- Disabled Netskope Support SSO

- Unlocked admin
- Edited SSO Admin Record
- Edit admin record
- Updated admin settings

Network Activity (4001) contains the following events:

Network Events are categorized using the record_type="network" and "action" fields. The supported "action" values are:

- Allow
- Block
- Bypass
- Closed
- Idle Timeout
- Proceed

All events with the value of /record_type = "connection" are also included.

Detection Finding (2004) contains the following events:

Alerts are categorized using the "alert_type" and "alert" fields, where the "alert" field value is set to "yes". The supported "alert_type" values are:

- Compromised Credential
- Malsite
- Malware
- Policy
- UBA
- C2

Data Security Finding (2006) contains the following events:

- /record_type = "alert" and /alert_type = "DLP"
- /record_type = "incident"

- /record_type = "epdlp" and /type = "endpoint"

File Hosting Activity (6006) contains the following events:

Application Events are categorized using the "activity" field, with the "alert" field value set to "no". The supported activity values are:

- Browser File Upload
- Attach
- Create
- Download
- Download All
- Download Installer
- Edit
- Quick Edit
- Insert
- Delete
- Delete All
- Copy
- Move
- Preview
- Formshare
- File Share Access
- Upload
- Share
- Post
- View
- Archive
- Bluetooth File Transfer
- Detach
- Print

- Publish

Device Inventory Info (5001) contains the following events:

All infrastructure logs with record_type="infrastructure".

CloudWatch pipelines configuration for Netskope

Collects security events and alerts from Netskope using REST API v2 token-based authentication.

Configure the Netskope source with the following parameters:

```
source:
  netskope_cloudexchange:
    tenant_url: "<<your-tenant-url>>"
    range: "P7D"
    authentication:
      api_token: "${{aws_secrets:netskope-account-credentials:apiToken}}"
```

Parameters

`tenant_url` (required)

Netskope tenant URL (for example, mycompany.goskope.com).

`authentication.api_token` (required)

REST API v2 token generated from the Netskope Service Account under RBACv3.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

Microsoft Entra ID integration configuration

Microsoft Entra ID (formerly Azure Active Directory) is Microsoft's cloud-based identity and access management service that helps organizations manage user identities and secure access to resources.

Topics

- Source configuration for Microsoft Entra ID

- CloudWatch pipelines configuration for Microsoft Entra ID

Source configuration for Microsoft Entra ID

Integrating with Microsoft Entra ID

Microsoft Entra ID (formerly Azure Active Directory) is Microsoft's cloud-based identity and access management service that helps organizations manage user identities and secure access to resources. CloudWatch Pipeline uses the Microsoft Graph API to retrieve comprehensive identity and security information from Microsoft Entra ID audit logs. The Microsoft Graph API provides access to three primary log types: Directory Audit Logs (tracking directory-level changes and administrative actions), Sign-In Logs (capturing user authentication events and activities), and Provisioning Logs (monitoring user and group provisioning operations).

Authenticating with Microsoft Entra ID

To retrieve the Audit Logs EntraID, pipelines needs to authenticate with your account. The plugin supports OAuth2 Authentication. Follow the instructions in Microsoft Graph APIs and should have the Microsoft Entra ID P1 or P2 license.

- Register an application in Azure with Supported account types, Accounts in this organizational directory only (Single tenant). After registration is complete, note down the Application (client) ID and Directory (tenant) ID.
- Generate a new key for your application. Key is also known as client secret, which are used when exchanging an authorization code for an access token.
- In the AWS Secrets Manager, create a secret and store the Application (client) ID under the key `client_id` and the client secret under the key `client_secret`
- Specify the permissions your application requires to access the Microsoft Graph APIs. The permissions you need are:
  - AuditLog.Read.All: Required to read audit logs, sign-in logs, and provisioning logs
  - Directory.Read.All: Required to read directory data

Configuring the CloudWatch Pipeline

When configuring the pipeline to read Audit Logs from Microsoft EntraID, choose Microsoft EntraID as the data source. Fill in the required information like Tenant Id using Directory (tenant) ID. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Entra ID events that map to Authentication (3002), Account Change (3001), User Access Management (3005), and Entity Management (3004).

Authentication contains the following events with type in brackets:

- Invalid Username or Password (Sign-in)
- User Strong Auth ClientAuthN Required Interrupt (Sign-in)
- Pass Through User Mfa Error (Sign-in)
- Authentication Failed During Strong Auth (Sign-in)

Account Change contains the following events with type in brackets:

- Add user (Audit)
- Update user (Audit)
- Delete user (Audit)
- Hard delete user (Audit)
- Reset password (Audit)
- User changed default security info (Audit)
- Enable Strong Authentication (Audit)
- Disable Strong Authentication (Audit)

User Access Management contains the following events with type in brackets:

- Add eligible member to role (Audit)
- Remove eligible member from role (Audit)
- Add eligible member to role in PIM completed (Audit)
- Remove eligible member from role in PIM completed (Audit)
- Add member to role (Audit)
- Remove member from role (Audit)
- Remove permanent direct role assignment (Audit)
- Add permanent direct role assignment (Audit)

- Triggered PIM alert (Audit)
- Add delegated permission grant (Audit)
- Remove delegated permission grant (Audit)

Entity Management contains the following events with type in brackets:

- Create (Provisioning)
- Update (Provisioning)
- Add app role assignment to service principal (Audit)
- Remove app role assignment to service principal (Audit)
- Add service principal credentials (Audit)
- Remove service principal credentials (Audit)
- Update service principal (Audit)
- Add service principal (Audit)
- Hard delete service principal (Audit)
- Remove service principal (Audit)
- Consent to application (Audit)
- Add application (Audit)
- Add owner to application (Audit)
- Hard Delete application (Audit)
- Delete application (Audit)
- Update application (Audit)
- Update application – Certificates and secrets management (Audit)
- Add device (Audit)
- Update device (Audit)
- Delete device (Audit)
- Hard delete device (Audit)

CloudWatch pipelines configuration for Microsoft Entra ID

Collects log data from Microsoft Entra ID (formerly Azure Active Directory) using OAuth2 authentication.

Configure the Microsoft Entra ID source with the following parameters:

```
source:
  microsoft_entraid:
    tenant_id: "<example-tenant-ID>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`tenant_id` (required)

The Microsoft Entra ID tenant ID for your organization.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Microsoft Graph API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Microsoft Graph API authentication.

Microsoft Windows Events integration configuration

Microsoft Windows Event Logs provide a comprehensive logging system that records system, security, and application events on Windows operating systems.

Topics

- Source configuration for Microsoft Windows Events
- CloudWatch pipelines configuration for Microsoft Windows Events

Source configuration for Microsoft Windows Events

Integrating with Windows Event

Microsoft Windows Event Logs provide a comprehensive logging system that records system, security, and application events on Windows operating systems. CloudWatch Pipeline uses the Log Analytics API to retrieve information about system operations, security events, user activities,

and application behaviors from Windows servers and workstations. The Log Analytics API enables access to event data through KQL (Kusto Query Language) queries, allowing retrieval of Windows Event logs from Log Analytics workspaces.

Authenticating with Windows Event

To read Windows Event audit Logs, the pipeline needs to authenticate with your account. The plugin supports OAuth2 Authentication. Follow these instructions to get started with Microsoft Windows Event: Log Analytics APIs.

- Register an application in Azure with Supported account types, Accounts in this organizational directory only (Single tenant). After registration is complete, note down the Application (client) ID and Directory (tenant) ID.
- Generate a new client secret for your application. The client secret is used when exchanging an authorization code for an access token. Copy the secret value immediately as it won't be shown again.
- In the AWS Secrets Manager, create a secret and store the Application (client) ID under the key `client_id` and the client secret under the key `client_secret`.
- Specify the API permissions your application requires to access the Log Analytics API. The permission you need is: Data.Read: Required to execute KQL queries and read log data from Log Analytics workspaces, including Windows Event logs.
- Create and configure a Log Analytics Workspace: Create a workspace in Azure portal (Monitor → Log Analytics workspaces). Create a Data Collection Rule (DCR) to specify which Windows Event Logs to collect (System, Application, Security). Connect your Windows servers/VMs to the workspace through the DCR. Note down your Workspace ID from the workspace Overview page (required for API queries)
- Grant workspace access to your application: Navigate to your Log Analytics workspace → Access control (IAM). Assign the Log Analytics Reader role to your registered application. This RBAC role works together with the API permission to provide secure access: OAuth confirms API usage rights, while IAM confirms workspace data access rights.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read logs, choose Microsoft Windows Events as the data source. Fill in the required information like Tenant Id using Directory (tenant) ID and Workspace Id (workspace_id). After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Windows audit events that map to Account Change (3001), Authentication (3002), Entity Management (3004), Event Log Activity (1008), File System Activity (1001), Group Management (3006), and Kernel Activity (1003).

Account Change contains the following events:

- 4740

Authentication contains the following events:

- 4624
- 4625
- 4634
- 4647
- 4648
- 4649
- 4672

Entity Management contains the following events:

- 4616
- 4907
- 4719
- 4902

Event Log Activity contains the following events:

- 1100
- 1102
- 1104
- 1105

File System Activity contains the following events:

- 4608
- 4660
- 4688
- 4696
- 4826
- 5024
- 5033
- 5058
- 5059
- 5061
- 5382
- 5379

Group Management contains the following events:

- 4732
- 4798
- 4799
- 4733
- 4731
- 4734
- 4735

Kernel Activity contains the following events:

- 4674

CloudWatch pipelines configuration for Microsoft Windows Events

Collects log data from Microsoft Windows Event logs using OAuth2 authentication.

Configure the Microsoft Windows Event source with the following parameters:

```
source:
```

```
  microsoft_windows:
    tenant_id: "<example-tenant-ID>"
    workspace_id: "<example-workspace-ID>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`tenant_id` (required)

The Microsoft tenant ID for your organization.

`workspace_id` (required)

The Microsoft Log Analytics workspace ID.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Log Analytics workspace API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Log Analytics workspace API authentication.

Okta Auth0

CloudWatch pipelines enables you to collect logs from Okta Auth0. Okta Auth0 is a flexible identity platform designed for modern application authentication and authorization.

Topics

- Source configuration for Okta Auth0
- Pipeline configuration for Okta Auth0

Source configuration for Okta Auth0

Integrating with Okta Auth0

Okta Auth0 is a flexible identity platform designed for modern application authentication and authorization. Auth0 provides developers with powerful tools to integrate secure login, user management, and access control into applications while maintaining scalability and customization.

CloudWatch Pipeline uses the Auth0 Management API to retrieve Authentication (successful and failed logins), and API Activity logs from Auth0 log events.

Authenticating with Okta Auth0

To read logs, the pipeline needs to authenticate with your Okta Auth0 tenant. Auth0 Management API access requires a Client ID and Client Secret belonging to a Machine-to-Machine (M2M) application.

Generate Client Credentials see API Settings for more details.

- Sign in to the Auth0 Dashboard using an admin account.
- Navigate to Applications → Applications.
- Select an existing Machine-to-Machine Application or create a new one.
- Make sure the application has the required scope permissions for the Management API, specifically: `read:logs`
- In the AWS Secrets Manager, create a secret and store the Client ID under the key `client_id` and the Client Secret under the key `client_secret`
- Identify your Auth0 Tenant Domain (for example: `yourtenant.us.auth0.com`) and give it in pipeline.

Once configured, the pipeline can authenticate using the Client Credentials flow and retrieve log events from Auth0.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose Okta Auth0 as the data source. Select the Source Type as Tenant and provide the required details such as your Auth0 Tenant Domain and Client Credentials. After you create the pipeline, log data from Okta Auth0 will be collected and made available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and the Auth0 events that maps to Authentication (3002) and API Activity (6003)

Authentication contains the following events:

- f

- fu
- fp
- feccft
- fepft
- feacft
- fc
- fco
- fcoa
- fd
- ferrt
- fertft
- fsa
- limit_wc
- limit_sul
- limit_mu
- pwd_leak
- reset_pwd_leak
- signup_pwd_leak
- gd_auth_fail_email_verification
- gd_auth_failed
- gd_auth_rejected
- gd_otp_rate_limit_exceed
- gd_recovery_failed
- gd_recovery_rate_limit_exceed
- gd_webauthn_challenge_failed
- passkey_challenge_failed
- scp
- sv
- ss
- s

- fi
- fv
- feoobft
- feotpft
- fercft
- ss_sso_failure
- fepotpft
- fvr
- flo

API Activity contains the following events:

- api_limit
- limit_delegation
- mgmt_api_read
- sapi
- api_limit_warning

Pipeline configuration for Okta Auth0

Collects log data from Okta Auth0 using OAuth2 authentication.

Configure the Okta Auth0 source with the following parameters:

```
source:
  okta_auth0:
    domain: "<example-domain-name>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`domain` (required)

The Okta Auth0 domain name for your organization.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Okta Auth0 API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Okta Auth0 API authentication.

Okta SSO integration configuration

CloudWatch pipelines enables you to collect logs from Okta SSO. Okta SSO is a leading cloud-based identity and access management solution that provides secure, centralized authentication for users and applications.

Topics

- Source configuration for Okta SSO
- CloudWatch pipelines configuration for Okta SSO

Source configuration for Okta SSO

Integrating with Okta SSO

CloudWatch Pipeline uses the Okta System Log API to retrieve Authentication, API Activity, Detection Finding and Entity Management events from your Okta SSO tenant.

Authenticating with Okta SSO

To read the logs, the pipeline needs to authenticate with your Okta SSO tenant. For Okta SSO, authentication is performed using the OAuth 2.0 Client Credentials (JWT Assertion) flow through an Okta API Services application.

Generate the private/public key pair for authentication

- Sign in to the Okta Admin Console using an administrator account.
- Navigate to Applications → Applications.
- Select an existing API Services Application or create a new one.
- Under General → Client Credentials, upload a public key or generate a new key. This key pair will be used to authenticate using a signed JWT assertion.

- Make sure the application has the required OAuth scopes assigned, specifically: `okta.logs.read`
- Admin Roles → Edit assignments → Role(Select Read-only Administrator)
- Copy the Client ID of the application.
- Store the client_id and client_secret(private key) in AWS Secrets Manager: `client_id` and `client_secret(private_key)` (the RSA private key used to sign the JWT assertion)
- Identify your Okta Organization URL and configure in the pipeline (for example: `https://yourdomain.okta.com`).

Once configured, the pipeline can authenticate using Okta's OAuth 2.0 Client Credentials (JWT Assertion) flow and begin retrieving audit log events from the Okta System Log API.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose Okta SSO as the data source. Fill in the required information like Okta Domain name. After you create and activate the pipeline, audit log data from Okta SSO will begin flowing into the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and Okta events that map to Authentication (3002), API Activity (6003), Detection Finding (2004), and Entity Management (3004).

Authentication contains the following events:

- user.authentication.auth
- user.authentication.auth_via_AD_agent
- user.authentication.auth_via_IDP
- user.authentication.auth_via_LDAP_agent
- user.authentication.auth_via_inbound_SAML
- user.authentication.auth_via_inbound_delauth
- user.authentication.auth_via_iwa
- user.authentication.auth_via_mfa
- user.authentication.auth_via_radius
- user.authentication.auth_via_richclient

- user.authentication.auth_via_social
- user.authentication.authenticate
- user.authentication.sso
- user.session.start
- user.session.impersonation.grant
- app.oauth2.signon
- user.session.impersonation.initiate
- user.authentication.universal_logout
- user.session.clear
- user.session.end
- user.authentication.slo
- user.authentication.universal_logout.scheduled
- user.session.expire
- user.session.impersonation.end
- user.authentication.verify
- policy.evaluate_sign_on
- user.mfa.attempt_bypass
- user.mfa.okta_verify
- user.mfa.okta_verify.deny_push
- user.mfa.okta_verify.deny_push_upgrade_needed
- user.mfa.factor.activate
- user.mfa.factor.deactivate
- user.mfa.factor.reset_all
- user.mfa.factor.suspend
- user.mfa.factor.unsuspend
- user.mfa.factor.update
- user.session.impersonation.extend
- user.session.impersonation.revoke
- user.session.access_admin_app

- user.session.context.change
- application.policy.sign_on.deny_access
- user.authentication.auth_unconfigured_identifier
- user.authentication.dsso_via_non_priority_source
- app.oauth2.invalid_client_credentials
- policy.auth_reevaluate.fail

API Activity contains the following events:

- oauth2.claim.created
- oauth2.scope.created
- security.trusted_origin.create
- system.api_token.create
- workflows.user.table.view
- app.oauth2.as.key.rollover
- app.saml.sensitive.attribute.update
- system.api_token.update
- oauth2.claim.updated
- oauth2.scope.updated
- security.events.provider.deactivate
- system.api_token.revoke
- oauth2.claim.deleted
- oauth2.scope.deleted

Detection Finding contains the following events:

- security.attack.start
- security.breached_credential.detected
- security.request.blocked
- security.threat.detected
- security.zone.make_blacklist

- system.rate_limit.violation
- user.account.report_suspicious_activity_by_enduser
- user.risk.change
- user.risk.detect
- zone.make_blacklist
- security.attack.end

Entity Management contains the following events:

- iam.role.create
- system.idp.lifecycle.create
- application.lifecycle.create
- group.lifecycle.create
- user.lifecycle.create
- policy.lifecycle.create
- zone.create
- oauth2.as.created
- event_hook.created
- inline_hook.created
- pam.security_policy.create
- iam.resourceset.create
- pam.secret.create
- analytics.reports.export.download
- app.audit_report.download
- system.idp.lifecycle.read_client_secret
- app.oauth2.client.read_client_secret
- pam.secret.reveal
- pam.service_account.password.reveal
- support.org.update
- system.idp.lifecycle.update

- application.lifecycle.update
- policy.lifecycle.update
- user.account.update_profile
- user.account.update_password
- user.account.reset_password
- group.profile.update
- zone.update
- group.privilege.grant
- group.privilege.revoke
- iam.resourceset.bindings.add
- user.account.privilege.grant
- user.account.privilege.revoke
- pki.cert.lifecycle.revoke
- iam.resourceset.update
- iam.role.update
- pam.security_policy.update
- oauth2.as.updated
- event_hook.updated
- inline_hook.updated
- pam.secret.update
- iam.resourceset.bindings.delete
- iam.role.delete
- pam.security_policy.delete
- policy.lifecycle.delete
- user.lifecycle.delete.initiated
- application.lifecycle.delete
- group.lifecycle.delete
- zone.delete
- oauth2.as.deleted
- event_hook.deleted

- inline_hook.deleted
- iam.resourceset.delete
- pam.secret.delete
- device.enrollment.create
- credential.register
- credential.revoke
- policy.lifecycle.activate
- system.feature.enable
- event_hook.activated
- inline_hook.activated
- system.feature.disable
- application.lifecycle.activate
- user.lifecycle.activate
- zone.activate
- oauth2.as.activated
- system.log_stream.lifecycle.activate
- policy.lifecycle.deactivate
- security.authenticator.lifecycle.deactivate
- application.lifecycle.deactivate
- user.lifecycle.deactivate
- zone.deactivate
- event_hook.deactivated
- inline_hook.deactivated
- system.log_stream.lifecycle.deactivate
- oauth2.as.deactivated
- user.account.lock
- user.account.lock.limit
- user.lifecycle.suspend
- device.lifecycle.suspend
- user.account.unlock

- user.lifecycle.unsuspend
- device.lifecycle.unsuspend
- user.lifecycle.reactivate

CloudWatch pipelines configuration for Okta SSO

Collects log data from Okta SSO using OAuth2 authentication.

Configure the Okta SSO source with the following parameters:

```
source:
  okta_sso:
    domain: "<example-domain-name>"
    range: "P7D"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`domain` (required)

The Okta domain name for your organization.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Okta SSO API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Okta SSO API authentication.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

OneLogin Identity integration configuration

OneLogin is a cloud-based identity and access management (IAM) platform that provides single sign-on (SSO), multi-factor authentication (MFA), and user provisioning capabilities.

Topics

- Source configuration for OneLogin Identity
- CloudWatch pipelines configuration for OneLogin Identity

Source configuration for OneLogin Identity

Integrating with OneLogin Identity

OneLogin is a cloud-based identity and access management (IAM) platform that provides single sign-on (SSO), multi-factor authentication (MFA), and user provisioning capabilities. CloudWatch pipelines uses the OneLogin Events API to retrieve information about authentication events, user activities, policy decisions, and administrative changes across your OneLogin environment. The Events API enables access to event data through REST endpoints, allowing retrieval of security and access logs from your OneLogin account.

Authenticating with OneLogin Identity

To read the logs, the pipeline needs to authenticate with your OneLogin account. For OneLogin, authentication is performed using OAuth2.

Configure OAuth2 authentication for OneLogin

- Log in to the OneLogin Admin Portal and navigate to Developers → API Credentials. Create a new API credential pair. Note the Client ID and Client Secret immediately.
- Assign the appropriate permissions. Select Read All or Manage All scope to make sure the credentials can access event log data.
- In AWS Secrets Manager, create a secret and store the Client ID under key `client_id` and the client secret under key `client_secret`.
- Note your Account ID (subdomain) from the OneLogin Admin Portal under Settings → Account Settings.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose OneLogin as the data source. Fill in the required information like subdomain and authentication credentials. Optionally, specify the Range duration format (for example, PT21H for the last 21 hours). After you create and activate the pipeline, event log data from OneLogin will begin flowing into the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and OneLogin events that map to Account Change (3001), Authentication (3002), and Entity Management (3004).

Account Change contains the following events:

- User requested new password
- Changed password for user
- User deactivated
- Password request approved from user
- User locked
- User suspended
- User locked out of app
- OTP device unlocked for user
- User suspended in app
- User suspended in directory
- Unlocked user in directory
- User granted permission to manage role
- User permission to manage role revoked
- User enabled desktop SSO
- User disabled desktop SSO
- Admin changed password for user
- Redirected to an external site for password reset
- API - password updated for user
- API - user locked
- User suspended through API
- User locked through API
- User enabled adaptive login for account
- User disabled adaptive login for account
- Profile change password
- Manually added user to app

- Manually removed user from app
- Failed to change password for user
- User granted permission to manage role failed
- User permission to manage role revoked failed
- Smart password updated for user
- Smart password could not be updated for user
- API - password not updated for user

Authentication contains the following events:

- User logged into OneLogin
- User logged out of OneLogin
- User logged into app
- User logged out of app
- User authenticated by RADIUS configuration
- User authenticated through API
- User successfully authenticated with VLDAP
- User signed in into OneLogin through social network
- User successfully authenticated with VLDAP (OneLogin Desktop Mac)
- API - user logged out
- API - verify factor called
- API - confirm OTP for user succeeded
- User was force logged out
- User successfully logged in on a trusted device
- User successfully logged in through OneLogin Desktop
- User denied auth through OTP push request
- User challenged for OTP
- User reauthenticated into app
- User verified OTP device
- OIDC password for app success

- API - trigger factor for user succeeded
- OIDC implicit flow for app success
- OIDC authorization code for app success
- OIDC get code for app success
- OIDC validate token for app success
- User failed authentication
- User failed to log into app
- User rejected by RADIUS configuration
- Failed to login to app through IDP
- Could not authenticate to app
- User failed authentication through API
- User failed authentication with VLDAP
- User authentication policy does not allow sign-in through social network
- User failed authentication with VLDAP (OneLogin Desktop Mac)
- API - user failed to log out
- API - verify factor failed
- API - confirm OTP for user failed
- User failed to log in on a trusted device
- User failed to login through OneLogin Desktop
- User failed to authenticate through OneLogin Desktop
- User failed OTP challenge
- OIDC implicit flow for app failed
- OIDC authorization code for app failed
- OIDC password for app failed
- OIDC validate token for app failed
- OIDC general fail
- OIDC get code for app failed

Entity Management contains the following events:

- Assigned role to user

- User was created
- User updated
- User deactivated
- User was activated
- User was deleted
- OTP device registered for user
- OTP device deregistered for user
- Updated credit card
- User provisioned in app
- User updated in app
- User suspended in app
- User reactivated in app
- User deleted in app
- Account granted permission to privilege
- Account revoked permission to privilege
- User granted permission to privilege
- User permission to privilege revoked
- Added trusted IDP
- Removed trusted IDP
- Modified trusted IDP
- User provisioned in directory
- User updated by directory
- User suspended in directory
- User reactivated in directory
- User deleted in directory
- Deleted secure note
- Updated user login information
- Attempted to update login information
- Changed the default trusted IDP

- User added to role
- User removed from role
- Created policy
- Updated policy
- Deleted policy
- Created proxy agent
- Deleted proxy agent
- Created RADIUS configuration
- Updated RADIUS configuration
- Deleted RADIUS configuration
- Enabled VPN
- Updated VPN settings
- Disabled VPN
- Enabled embedding
- Updated embedding settings
- Disabled embedding
- Created authentication factor
- Updated authentication factor
- Deleted authentication factor
- Updated security questions
- Updated desktop SSO settings
- Enabled desktop SSO
- Disabled desktop SSO
- Created certificate
- Deleted certificate
- Created API credential
- Deleted API credential
- Enabled API credential
- Disabled API credential

- Enabled virtual LDAP
- Disabled virtual LDAP
- Updated virtual LDAP settings
- Enabled branding
- Disabled branding
- Updated branding
- Deleted mapping
- Disabled mapping
- Enabled mapping
- Updated mapping
- Deleted custom user fields
- Updated company info
- Updated account settings
- Deleted directory
- Deleted connector instance from directory
- Created self registration
- Updated self registration
- Deleted self registration
- Created payment record
- Updated payment record
- Deleted payment record
- Updated terms and conditions for policy
- Manually updated user login for app
- User was created by trusted IDP
- Directory external ID was updated for user
- Directory external ID was deleted for user
- Updated broadcaster
- Deleted broadcaster
- API - roles added to user
- API - roles removed for user

- API - user updated
- API - user deleted
- API - user created
- Updated directory
- OUs were updated for directory
- User suspended through API
- User reactivated through API
- App was updated
- Connector was created
- Connector was updated
- Connector was deleted
- Parameter was created
- Parameter was updated
- Parameter was deleted
- Deleted device for OneLogin Desktop
- Revoked user certificate
- Revoked device certificate
- App was created through API
- App was updated through API
- App was destroyed through API
- Sandbox deleted
- Sandbox created
- Sandbox updated
- User deleted security factor
- User renamed security factor
- Created RADIUS attribute
- Updated RADIUS attribute
- Deleted RADIUS attribute
- Role created
- Role deleted

- SMTP configuration updated
- Smart hook created
- Smart hook updated
- Smart hook deleted
- Smart hook environment variable created
- Smart hook environment variable updated
- Smart hook environment variable deleted
- API - privilege was created
- Created privilege
- API - privilege was updated
- Updated privilege
- API - privilege was deleted
- Deleted privilege
- API - privilege was assigned to user
- Assigned privilege to user
- API - privilege removed from user
- Removed privilege from user
- API - privilege assigned to role
- Assigned privilege to role
- API - privilege removed from role
- Removed privilege from role
- Report created
- Report updated
- Report destroyed
- Created group
- Updated group
- Destroyed group
- Created secure note
- API - app rules create success
- API - app rules update success

- API - app rules delete success
- API - roles update success
- Credit card update failed
- User could not be updated
- User could not be deleted in app
- User could not be updated in app
- User not updated in app
- API - user not deleted
- API - user not updated
- API - user not created
- Connector could not be created
- Connector could not be updated
- Connector could not be deleted
- Parameter could not be created
- Parameter could not be updated
- Parameter could not be deleted
- App failed to create through API
- App failed to update through API
- App failed to destroy through API
- Failed to delete sandbox
- Failed to create sandbox
- Failed to update sandbox
- Smart hook update failed
- Smart hook environment variable update failed
- API - app rules create failed
- API - app rules update failed
- API - app rules delete failed
- User added to role failed
- Role created failed
- Role deleted failed

- API - roles update failed

CloudWatch pipelines configuration for OneLogin Identity

Collects event logs from OneLogin using OAuth2 authentication.

Configure the OneLogin Identity source with the following parameters:

```
source:
  onelogin_identity:
    range: "P7D"
    subdomain: "<your-subdomain>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`range` (optional)

For pulling historical logs. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for the last 21 hours). The default is 0 hours, and the maximum is 90 days.

`subdomain` (required)

Your OneLogin account subdomain. Must be alphanumeric with hyphens only, between 1–35 characters.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for OneLogin Events API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for OneLogin Events API authentication.

> **Note**
>
> The `client_id` and `client_secret` values are retrieved from AWS Secrets Manager. The above parameter information can be obtained from the API credentials generated while setting up your OneLogin application.

Palo Alto Networks Next-Generation Firewalls integration configuration

CloudWatch pipelines enables you to collect network security logs from Palo Alto Networks Next-Generation Firewalls. Palo Alto Networks provides network security solutions, including firewall, VPN, and threat detection services.

Topics

- Source configuration for Palo Alto Networks Next-Generation Firewalls
- CloudWatch pipelines configuration for Palo Alto Networks Next-Generation Firewalls

Source configuration for Palo Alto Networks Next-Generation Firewalls

Integrating with Palo Alto Networks Next-Generation Firewalls

CloudWatch Pipeline integrates with Palo Alto Networks NGFW using the PAN-OS XML API to retrieve security, authentication, network activity, process activity, detection finding and threat activity. The PAN-OS XML API enables structured access, allowing the retrieval of System Logs, GlobalProtect, Traffic Logs, Threat Logs and URL Filtering Log.

Authenticating with Palo Alto NGFW

To read network security logs, the pipeline needs to authenticate with your Palo Alto Networks NGFW login device interface. The plugin supports Basic Authentication.

- Create and Manage Users on a Palo Alto Networks NGFW Firewall by using the CLI
- Login firewall using with hostname using user admin and your password
- Store this username and password in a secret in AWS Secrets Manager under the keys `username` and `password`.
- Identify and note down your PAN-OS hostname.

Once configured, the pipeline can authenticate using the username and password and retrieve log activity from PAN-OS.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read logs from Palo Alto Networks NGFW, choose Palo Alto Networks Next-Generation Firewalls as the data source. Fill in the required information like

`hostname`. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to Authentication (3002), Network Activity (4001), Process Activity (1007), and Detection Finding (2004).

Authentication contains the following type and subtypes:

- GlobalProtect
  - data
  - file
  - flood
  - packet
  - scan
  - spyware
  - url
  - virus
  - vulnerability
  - wildfire
  - wildfire-virus
- System Logs
  - auth

Network Activity contains the following types and subtypes:

- Traffic Logs
  - start
  - end
  - drop
  - deny
- System Logs
  - vpn

- url-filtering
- app-cloud-engine
- dhcp
- ssh
- dnsproxy
- dns-security
- wildfire
- wildfire-appliance
- ntpd
- userid

Process Activity contains the following type and subtypes:

- System Logs
  - general
  - satd
  - ras
  - sslmgr
  - hw
  - iot
  - ctd-agent
  - routing
  - port
  - device-telemetry

Detection Finding contains the following type and subtypes:

- Threat Logs
  - data
  - file
  - flood
  - packet

- scan
- spyware
- url
- ml-virus
- virus
- vulnerability
- wildfire
- wildfire-virus

- URL Filtering Log

CloudWatch pipelines configuration for Palo Alto Networks Next-Generation Firewalls

Collects log data from Palo Alto Next-Generation Firewall using basic authentication.

Configure the Palo Alto NGFW source with the following parameters:

```
source:
  paloaltonetworks_nextgenerationfirewall:
    hostname: "<example-host-name>"
    authentication:
      basic:
        username: "${{aws_secrets:<secret-name>:username}}"
        password: "${{aws_secrets:<secret-name>:password}}"
```

Parameters

`hostname` (required)

The Palo Alto NGFW hostname for your firewall.

`authentication.basic.username` (required)

Basic authentication username for Palo Alto NGFW API authentication.

`authentication.basic.password` (required)

Basic authentication password for Palo Alto NGFW API authentication.

Palo Alto Prisma Cloud integration configuration

Palo Alto Prisma Cloud is a cloud-native application protection platform (CNAPP) that secures cloud infrastructure, workloads, and applications across multi-cloud environments. CloudWatch pipelines use the Prisma Cloud CSPM and Compute (CWPP) APIs to retrieve security alerts, vulnerability findings, asset inventories, compliance posture, audit logs, and workload data from your Prisma Cloud tenant.

Topics

- Source configuration for Palo Alto Prisma Cloud
- CloudWatch pipelines configuration for Palo Alto Prisma Cloud

Source configuration for Palo Alto Prisma Cloud

Integrating with Palo Alto Prisma Cloud

CloudWatch pipelines use the Prisma Cloud CSPM and Compute (CWPP) APIs to retrieve security alerts, vulnerability findings, asset inventories, compliance posture, audit logs, and workload data from your Prisma Cloud tenant. The APIs expose REST endpoints that allow fetching security data for monitoring and analysis.

To integrate CloudWatch Pipelines with Palo Alto Prisma Cloud, complete the following high-level steps:

- Generate an Access Key and Secret Key in the Prisma Cloud Console.
- Identify the CSPM API URL, Compute Console URL, and Compute API version.
- Store the credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with Palo Alto Prisma Cloud as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- An active Palo Alto Prisma Cloud Enterprise tenant with API access
- A Prisma Cloud user with System Admin or Account Group Read Only role

- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager
- An AWS account with permissions to create and manage CloudWatch Logs log groups

Authenticating with Palo Alto Prisma Cloud

Palo Alto Prisma Cloud uses an Access Key and Secret Key for authentication. The pipeline sends these credentials to the CSPM login endpoint, which returns a JWT token for subsequent API access.

Configure Authentication for Palo Alto Prisma Cloud

To configure authentication credentials for the pipeline:

1. Log in to the Prisma Cloud Console and navigate to Settings > Access Control > Access Keys.
2. Choose **Add** > **Access Key**.
3. Note the Access Key ID and Secret Key.
4. Identify the CSPM API URL based on your tenant region (for example, `https://api.prismacloud.io`).
5. Identify the Compute Console URL by navigating to Compute > Manage > System > Utilities > Path to Console.
6. Identify the Compute API version (available from the bell icon in the Console).
7. Store the Access Key ID as `username` and Secret Key as `password` in AWS Secrets Manager.

Configuring the CloudWatch Pipeline

To configure the pipeline, choose Palo Alto Prisma Cloud as the data source. Provide the `cspm_base_url`, `compute_console_url`, `compute_api_version`, `access_key`, and `secret_key`. After you create and activate the pipeline, security data from Prisma Cloud will begin flowing into the selected CloudWatch Logs log group.

The following optional parameters are available:

- **Page sizes** – Configure the number of records retrieved per API request page.
- **Polling intervals** – Valid range is 10 minutes to 7 days. Uses ISO 8601 duration format.

- **Backfill durations** – Valid range is 1 hour to 90 days. Uses ISO 8601 duration format.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports the following OCSF event classes:

- **Base Event [0]** – Applications
- **Detection Finding [2004]** – Alerts
- **Vulnerability Finding [2002]** – Vulnerabilities
- **Cloud Resources Inventory Info [5023]** – Assets
- **Entity Management [3004]** – Audit Logs
- **Compliance Finding [2003]** – Compliances, Alerts
- **Device Inventory Info [5001]** – Hosts
- **Scan Activity [6007]** – Images, Containers

CloudWatch pipelines configuration for Palo Alto Prisma Cloud

Collects security data from Palo Alto Prisma Cloud using Access Key and Secret Key authentication through the CSPM and Compute APIs.

Configure the Palo Alto Prisma Cloud source with the following parameters:

```
source:
  paloaltonetworks_prismacloud:
    cspm_base_url: "https://api.prismacloud.io"
    compute_console_url: "https://us-east1.cloud.twistlock.com/us-1-123456789"
    compute_api_version: "33.01"
    authentication:
      oauth2:
        access_key: "${{aws_secrets:my-prismacloud-secret:username}}"
        secret_key: "${{aws_secrets:my-prismacloud-secret:password}}"
    range: "P30D"
```

Parameters

`cspm_base_url` (required)

The base URL for the Prisma Cloud CSPM API, determined by your tenant region (for example, `https://api.prismacloud.io`).

`compute_console_url` (required)

The URL of the Prisma Cloud Compute Console (for example, `https://us-east1.cloud.twistlock.com/us-1-123456789`).

`compute_api_version` (required)

The Compute API version for your Prisma Cloud tenant (for example, `33.01`).

`authentication.oauth2.access_key` (required)

The Prisma Cloud Access Key ID, stored in AWS Secrets Manager.

`authentication.oauth2.secret_key` (required)

The Prisma Cloud Secret Key, stored in AWS Secrets Manager.

`range` (optional)

The historical time period for backfilling data. Uses ISO 8601 duration format. Minimum is `PT1H`, maximum is `P90D`. Default is `P30D`.

Ping Identity PingAccess integration configuration

Ping Identity PingAccess is an enterprise-grade access management solution that serves as a centralized policy enforcement point for web applications and APIs. It provides fine-grained authorization, URL-level access control, and web session management across enterprise resources. CloudWatch pipelines allow ingestion of Ping Identity PingAccess audit log data into CloudWatch Logs.

Topics

- Source configuration for Ping Identity PingAccess
- CloudWatch pipelines configuration for Ping Identity PingAccess

Source configuration for Ping Identity PingAccess

Integrating with Ping Identity PingAccess

Ping Identity PingAccess integration uses Amazon S3 and Amazon SQS to ingest audit log data into CloudWatch pipelines. PingAccess is deployed on-premises and does not expose a REST API for audit log retrieval. Fluent Bit tails the JSON-formatted audit log files on the PingAccess host and delivers them to an Amazon S3 bucket. Amazon SQS notifications alert the pipeline when new log objects arrive.

Prerequisites

- An AWS account with permissions to create Amazon S3 buckets, Amazon SQS queues, and IAM roles
- A Ping Identity PingAccess server with administrative access
- A Linux host with Fluent Bit installed (can be the same host as PingAccess)
- Network connectivity between the PingAccess host and AWS

Log forwarding setup

PingAccess writes audit logs to local files using Log4j2. You configure JSON-formatted output by adding JsonTemplateLayout appenders to the `log4j2.xml` configuration file. Fluent Bit tails these JSON log files and delivers them to Amazon S3. For detailed configuration instructions, see Step 5 in the following section.

Instructions to setup Amazon S3 and Amazon SQS

Complete the following steps to configure the Amazon S3 and Amazon SQS infrastructure for Ping Identity PingAccess log ingestion.

Step 1: Create Amazon S3 bucket

Create an Amazon S3 bucket to store PingAccess audit logs. The bucket must reside in the same AWS Region where you plan to create the CloudWatch pipeline.

Step 2: Create Amazon SQS queue

Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue receives notifications when new log files are added to the bucket.

Step 3: Connect Amazon S3 to Amazon SQS

Configure the Amazon S3 bucket to send event notifications for `s3:ObjectCreated:*` events to the Amazon SQS queue.

Step 4: Configure Amazon SQS queue policy

Configure the Amazon SQS queue policy to allow the Amazon S3 bucket to send messages to the queue. Apply the following policy to your Amazon SQS queue:

```
{
```

```
  "Version": "2012-10-17",
  "Id": "AllowS3ToSQS",
  "Statement": [
    {
      "Sid": "AllowS3BucketNotification",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "<account-id>"
        }
      }
    }
  ]
}
```

Step 5: Configure Ping Identity PingAccess log export and Fluent Bit

Enable JSON logging on PingAccess by editing <PA_HOME>/conf/log4j2.xml. Add the following RollingFile appenders:

```
<RollingFile name="AdminAudit-JsonTemplateLayout"
             fileName="${sys:pa.log.dir}/admin_audit_json.log"
             filePattern="${sys:pa.log.dir}/admin_audit_json.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
    <JsonTemplateLayout eventTemplateUri="${sys:pa.log4j.json.templates.uri}/admin-
audit-log.json"/>
    <Policies>
        <TimeBasedTriggeringPolicy maxRandomDelay="45"/>
    </Policies>
</RollingFile>

<RollingFile name="AccessAudit-JsonTemplateLayout"
             fileName="${sys:pa.log.dir}/audit_json.log"
             filePattern="${sys:pa.log.dir}/audit_json.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
```

```
    <JsonTemplateLayout eventTemplateUri="${sys:pa.log4j.json.templates.uri}/access-
audit-log.json"/>
    <Policies>
        <TimeBasedTriggeringPolicy maxRandomDelay="45"/>
    </Policies>
</RollingFile>
```

Update the Loggers section to reference the new appenders:

```
<Logger name="com.pingidentity.pa.admin.audit" level="INFO" additivity="false">
    <AppenderRef ref="AdminAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="com.pingidentity.pa.audit.AccessAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="AccessAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="com.pingidentity.pa.audit.PolicyAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="AccessAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="com.pingidentity.pa.audit.SessionAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="AccessAudit-JsonTemplateLayout"/>
</Logger>
```

Configure Fluent Bit to collect both audit log files and deliver them to Amazon S3:

```
########################################
# SERVICE SECTION
########################################
[SERVICE]
    Flush        5
    Daemon       Off
    Log_Level    info
    storage.path /etc/fluent-bit/db/storage
    storage.sync normal
    storage.checksum off
    storage.backlog.mem_limit 10M

########################################
# INPUT SECTION - Access Audit Log
########################################
[INPUT]
```

```
    Name              tail
    Path              /opt/pingaccess/log/audit_json.log
    Tag               pingaccess.audit
    Read_from_Head    True
    DB                /etc/fluent-bit/db/audit_checkpoint.db
    DB.Sync           Normal
    storage.type      filesystem
    Rotate_Wait       30
    Refresh_Interval 60

########################################
# INPUT SECTION - Admin Audit Log
########################################
[INPUT]
    Name              tail
    Path              /opt/pingaccess/log/admin_audit_json.log
    Tag               pingaccess.admin
    Read_from_Head    True
    DB                /etc/fluent-bit/db/admin_checkpoint.db
    DB.Sync           Normal
    storage.type      filesystem
    Rotate_Wait       30
    Refresh_Interval 60

########################################
# OUTPUT SECTION
########################################
[OUTPUT]
    Name               s3
    Match              pingaccess.*
    Bucket             amzn-s3-demo-bucket
    Region             <region>
    S3_Key_Format      /pingaccess/%Y/%m/%d/%H/%M/%S.log.gz
    compression        gzip
    use_put_object     On
    store_dir          /etc/fluent-bit/db/s3
    total_file_size    1M
    log_key            log
    upload_timeout     60
```

> **Note**
>
> PingAccess writes independent log files per cluster node. Fluent Bit must be installed on each node.

The `audit_json.log` captures runtime access control events (request URL, client IP, policy decision, response code, latency). The `admin_audit_json.log` captures administrative console activity (configuration changes, admin logins, rule and application modifications).

Fluent Bit delivers gzip-compressed NDJSON files to Amazon S3. The `storage.type filesystem` setting ensures buffering to disk for reliability.

> **Important**
>
> Use only the default PingAccess-provided JSON templates.

Step 6: IAM permissions

Create an IAM policy with the following permissions for the Fluent Bit host to write objects to the Amazon S3 bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FluentBitS3Write",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/pingaccess/*"
      ]
    }
```

```
  ]
}
```

Step 7: Verify

Verify the setup by confirming that log files appear in the Amazon S3 bucket and that Amazon SQS notifications are being generated. Check the Amazon SQS queue for messages indicating new object creation events.

Configuring the CloudWatch pipeline

- Choose Ping Identity PingAccess as the data source when creating the pipeline.
- Provide the Amazon SQS queue URL and IAM role ARN.
- Select the destination CloudWatch Logs log group.
- After you create the pipeline, data will be available in the selected log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. The following table shows the mapping between PingAccess audit log types and OCSF event classes.

Event name	OCSF event class
Access Audit (runtime HTTP activity)	HTTP Activity [4002]
API Audit (admin API operations)	API Activity [6003]

CloudWatch pipelines configuration for Ping Identity PingAccess

The Ping Identity PingAccess setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
```

```
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "pingidentity_pingaccess"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. Use "pingidentity_pingaccess" for Ping Identity PingAccess.

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Use `ndjson` for PingAccess JSON audit logs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

Ping Identity PingFederate integration configuration

Ping Identity PingFederate is an enterprise-grade identity federation server that serves as a central broker for Single Sign-On (SSO), token issuance, and protocol translation across SAML 2.0, OAuth 2.0, OpenID Connect, WS-Federation, and WS-Trust. CloudWatch pipelines allow ingestion of Ping Identity PingFederate audit log data into CloudWatch Logs.

Topics

- Source configuration for Ping Identity PingFederate
- CloudWatch pipelines configuration for Ping Identity PingFederate

Source configuration for Ping Identity PingFederate

Integrating with Ping Identity PingFederate

Ping Identity PingFederate integration uses Amazon S3 and Amazon SQS to ingest audit log data into CloudWatch pipelines. PingFederate is deployed on-premises and does not expose a REST API for audit log retrieval. Fluent Bit tails the JSON-formatted audit log files on the PingFederate host and delivers them to an Amazon S3 bucket. Amazon SQS notifications alert the pipeline when new log objects arrive.

Prerequisites

- An AWS account with permissions to create Amazon S3 buckets, Amazon SQS queues, and IAM roles
- A Ping Identity PingFederate server with administrative access
- A Linux host with Fluent Bit installed (can be the same host as PingFederate)
- Network connectivity between the PingFederate host and AWS

Log forwarding setup

PingFederate writes audit logs to local files using Log4j2. You configure JSON-formatted output by adding JsonTemplateLayout appenders to the `log4j2.xml` configuration file. Fluent Bit tails these JSON log files and delivers them to Amazon S3. For detailed configuration instructions, see Step 5 in the following section.

Instructions to setup Amazon S3 and Amazon SQS

Complete the following steps to configure the Amazon S3 and Amazon SQS infrastructure for Ping Identity PingFederate log ingestion.

Step 1: Create Amazon S3 bucket

Create an Amazon S3 bucket to store PingFederate audit logs. The bucket must reside in the same AWS Region where you plan to create the CloudWatch pipeline.

Step 2: Create Amazon SQS queue

Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue receives notifications when new log files are added to the bucket.

Step 3: Connect Amazon S3 to Amazon SQS

Configure the Amazon S3 bucket to send event notifications for `s3:ObjectCreated:*` events to the Amazon SQS queue.

Step 4: Configure Amazon SQS queue policy

Configure the Amazon SQS queue policy to allow the Amazon S3 bucket to send messages to the queue. Apply the following policy to your Amazon SQS queue:

```
{
  "Version": "2012-10-17",
  "Id": "AllowS3ToSQS",
  "Statement": [
    {
      "Sid": "AllowS3BucketNotification",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "StringEquals": {
          "aws:SourceAccount": "<account-id>"
```

```
        }
      }
    }
  ]
}
```

Step 5: Configure Ping Identity PingFederate log export and Fluent Bit

Enable JSON logging on PingFederate by editing <PF_HOME>/server/default/conf/log4j2.xml. Add the following RollingFile appenders:

```
<RollingFile name="AdminAudit-JsonTemplateLayout"
             fileName="${sys:pf.log.dir}/admin_json.log"
             filePattern="${sys:pf.log.dir}/admin_json.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
    <JsonTemplateLayout eventTemplateUri="${sys:pf.log4j.json.templates.uri}/admin-
audit-log.json"/>
    <Policies>
        <TimeBasedTriggeringPolicy maxRandomDelay="45"/>
    </Policies>
</RollingFile>

<RollingFile name="SecurityAudit-JsonTemplateLayout"
             fileName="${sys:pf.log.dir}/audit_json.log"
             filePattern="${sys:pf.log.dir}/audit_json.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
    <JsonTemplateLayout eventTemplateUri="${sys:pf.log4j.json.templates.uri}/runtime-
audit-log.json"/>
    <Policies>
        <TimeBasedTriggeringPolicy maxRandomDelay="45"/>
    </Policies>
</RollingFile>
```

Update the Loggers section to reference the new appenders for the multiple audit loggers:

```
<Logger name="org.sourceid.websso.profiles.sp.SpAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="org.sourceid.websso.profiles.idp.IdpAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
```

```
<Logger name="org.sourceid.websso.profiles.idp.AsAuditLogger" level="INFO"
 additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="org.sourceid.websso.profiles.idp.ClientRegistrationAuditLogger"
 level="INFO" additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="org.sourceid.wstrust.log.STSAuditLogger" level="INFO" additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="com.pingidentity.sdk.logging.LoggingUtil" level="INFO"
 additivity="false">
    <AppenderRef ref="SecurityAudit-JsonTemplateLayout"/>
</Logger>
<Logger name="AuditLogger" level="INFO" additivity="false">
    <AppenderRef ref="AdminAudit-JsonTemplateLayout"/>
</Logger>
```

Configure Fluent Bit to collect both audit log files and deliver them to Amazon S3:

```
########################################
# SERVICE SECTION
########################################
[SERVICE]
    Flush        5
    Daemon       Off
    Log_Level    info
    storage.path /etc/fluent-bit/db/storage
    storage.sync normal
    storage.checksum off
    storage.backlog.mem_limit 10M

########################################
# INPUT SECTION - Audit Log
########################################
[INPUT]
    Name             tail
    Path             /opt/pingfederate/log/audit_json.log
    Tag              pingfederate.audit
    Read_from_Head   True
    DB               /etc/fluent-bit/db/audit_checkpoint.db
    DB.Sync          Normal
```

```
    storage.type     filesystem
    Rotate_Wait      30
    Refresh_Interval 60

########################################
# INPUT SECTION - Admin Log
########################################
[INPUT]
    Name             tail
    Path             /opt/pingfederate/log/admin_json.log
    Tag              pingfederate.admin
    Read_from_Head   True
    DB               /etc/fluent-bit/db/admin_checkpoint.db
    DB.Sync          Normal
    storage.type     filesystem
    Rotate_Wait      30
    Refresh_Interval 60

########################################
# OUTPUT SECTION
########################################
[OUTPUT]
    Name              s3
    Match             pingfederate.*
    Bucket            amzn-s3-demo-bucket
    Region            <region>
    S3_Key_Format     /pingfederate/%Y/%m/%d/%H/%M/%S.log.gz
    compression       gzip
    use_put_object    On
    store_dir         /etc/fluent-bit/db/s3
    total_file_size   1M
    log_key           log
    upload_timeout    60
```

Note

PingFederate writes independent log files per cluster node. You must install Fluent Bit on each node.

- The `audit_json.log` file captures runtime authentication events (SSO, credential validation, assertion issuance).

- The `admin_json.log` file captures administrative console activity (configuration changes, admin logins).
- Fluent Bit delivers gzip-compressed NDJSON files to Amazon S3.
- The `storage.type filesystem` setting ensures buffering to disk for reliability.

> **Important**
>
> Use only the default PingFederate-provided JSON templates. Modifying the `.json` template files changes the output schema and might break downstream processing.

Step 6: IAM permissions

Create an IAM policy with the following permissions for the Fluent Bit host to write objects to the Amazon S3 bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FluentBitS3Write",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/pingfederate/*"
      ]
    }
  ]
}
```

Step 7: Verify

Verify the setup by confirming that log files appear in the Amazon S3 bucket and that Amazon SQS notifications are being generated. Check the Amazon SQS queue for messages indicating new object creation events.

Configuring the CloudWatch pipeline

- Choose Ping Identity PingFederate as the data source when creating the pipeline.
- Provide the Amazon SQS queue URL and IAM role ARN.
- Select the destination CloudWatch Logs log group.
- After you create the pipeline, data will be available in the selected log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0. The following table shows the mapping between PingFederate audit log types and OCSF event classes.

Event name	OCSF event class
Security Audit (runtime authentication lifecycle)	Authentication [3002]
Administrator Audit (admin login/logout)	Authentication [3002]
Administrator Audit (CRUD on configuration objects)	Entity Management [3004]

CloudWatch pipelines configuration for Ping Identity PingFederate

The Ping Identity PingFederate setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    aws:
```

```
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "pingidentity_pingfederate"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. Use "pingidentity_pingfederate" for Ping Identity PingFederate.

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Use `ndjson` for PingFederate JSON audit logs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

PingIdentity PingOne integration configuration

PingOne is a cloud-based identity and access management (IAM) platform offered by Ping Identity, designed to provide secure access to applications and services across various platforms.

Topics

- Source configuration for PingIdentity PingOne
- CloudWatch pipelines configuration for PingIdentity PingOne

Source configuration for PingIdentity PingOne

Integrating with PingIdentity PingOne

PingOne is Ping Identity's cloud-based identity-as-a-service (IDaaS) platform that provides identity and access management capabilities. CloudWatch Pipeline uses the PingOne Audit Logs API to retrieve information about authentication events, user activities, policy decisions, and administrative changes across your PingOne environment. The Audit Logs API enables access to event data through REST endpoints, allowing retrieval of security and access logs from your PingOne organization.

Authenticating with PingIdentity PingOne

To read the logs, the pipeline needs to authenticate with your PingOne environment. For PingOne, authentication is performed using OAuth2.

Configure OAuth2 authentication for PingOne

- Log in to the PingOne Console and navigate to Applications → Applications. Create a new application of type Worker. Note the Client ID and Environment ID.
- Generate a new Client Secret from the Configuration tab. Copy the secret immediately.
- In AWS Secrets Manager, create a secret and store the Client ID under key `client_id` and the client secret under key `client_secret`.
- Assign Environment Admin and Application Owner roles to the application.
- Identify your PingOne Region (NA, EU, AP, AU, CA, SG).
- Note the Environment ID from Settings → Environment → Properties.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose PingOne as the data source. Fill in the required information like Environment ID. Optionally, specify the Region (defaults to NA) and the Range duration format (for example, PT21H for the last 21 hours). The default range is 0 hours, and the maximum is 90 days. After you create and activate the pipeline, audit log data from PingOne will begin flowing into the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and PingOne events that map to Account Change (3001), Authentication (3002), and Entity Management (3004).

Account Change contains the following events:

- USER.CREATED
- USER.INVITED
- USER.REINVITED
- USER.INVITE_ACCEPTED
- PASSWORD.FORCE_CHANGE
- PASSWORD.RECOVERY
- PASSWORD.RESET
- USER.INVITE_REVOKED
- USER.DELETED
- USER.LOCKED
- MFA_SETTINGS.UPDATED
- PASSWORD.UNLOCKED
- USER.UNLOCKED

Authentication contains the following events:

- AUTHENTICATION.CREATED
- RADIUS_SESSION.CREATED
- SESSION.CREATED
- SESSION.UPDATED

- SESSION.DELETED
- USER.SLO_FAILURE
- USER.SLO_PARTIAL_LOGOUT
- USER.SLO_REQUESTED
- USER.SLO_SUCCESS
- USER.KERBEROS_FAILED
- USER.KERBEROS_SUCCEEDED
- DEVICE.ACTIVATION_OTP_FAILED
- DEVICE.ACTIVATION_OTP_INVALID
- DEVICE_PAYLOAD.CHECK_INVALID
- DEVICE_PAYLOAD.CHECK_SUCCESS
- OTP.CHECK_FAILED
- OTP.CHECK_INVALID
- OTP.CHECK_SUCCESS
- PASSWORD.CHECK_FAILED
- PASSWORD.CHECK_SUCCEEDED

Entity Management contains the following events:

- ACTION.CREATED
- AGREEMENT.CREATED
- AGREEMENT_LANGUAGE.CREATED
- AGREEMENT_LANGUAGE_REVISION.CREATED
- APPLICATION.CREATED
- AUTHORIZE_POLICY.CREATED
- CERTIFICATE.CREATED
- DEVICE.CREATED
- DEVICE_AUTHENTICATION_POLICY.CREATED
- FIDO_POLICY.CREATED
- FLOW.CREATED
- FLOW_DEFINITION.CREATED

- FLOW_EXECUTION.CREATED
- GROUP.CREATED
- IDENTITY_PROVIDER.CREATED
- IDP_ATTRIBUTE.CREATED
- INSTANT_MESSAGING_DELIVERY_SETTINGS.CREATED
- KEY.CREATED
- LICENSE.CREATED
- NOTIFICATION.CREATED
- NOTIFICATION_POLICY.CREATED
- ORGANIZATION.CREATED
- POLICY.CREATED
- RISK_POLICY_SET.CREATED
- SAML_ATTRIBUTE.CREATED
- SCHEMA_ATTRIBUTE.CREATED
- SIGN_ON_POLICY_ASSIGNMENT.CREATED
- VERIFY_POLICY.CREATED
- CERTIFICATE.READ
- KEY.READ
- SECRET.READ
- ACTION.UPDATED
- ADMIN_CONFIGURATION.UPDATED
- AGREEMENT.UPDATED
- AGREEMENT_LANGUAGE.UPDATED
- AGREEMENT_LANGUAGE_REVISION.UPDATED
- APPLICATION.UPDATED
- AUTHORIZE_POLICY.UPDATED
- CERTIFICATE.UPDATED
- DEVICE.NICKNAME_UPDATED
- DEVICE.UPDATED
- DEVICE_AUTHENTICATION_POLICY.UPDATED

- FIDO_POLICY.UPDATED
- FLOW.UPDATED
- FLOW_DEFINITION.UPDATED
- FLOW_EXECUTION.UPDATED
- GROUP.UPDATED
- IDENTITY_PROVIDER.UPDATED
- IDP_ATTRIBUTE.UPDATED
- INSTANT_MESSAGING_DELIVERY_SETTINGS.UPDATED
- KEY.UPDATED
- LICENSE.UPDATED
- NOTIFICATION.UPDATED
- NOTIFICATION_POLICY.UPDATED
- NOTIFICATIONS_SETTINGS.UPDATED
- ORGANIZATION.UPDATED
- POLICY.UPDATED
- RISK_POLICY_SET.ORDER_UPDATED
- RISK_POLICY_SET.UPDATED
- SAML_ATTRIBUTE.UPDATED
- SCHEMA_ATTRIBUTE.UPDATED
- SECRET.UPDATED
- SETTINGS.UPDATED
- SIGN_ON_POLICY_ASSIGNMENT.UPDATED
- USER.QUOTA_RESET
- USER.UPDATED
- VERIFY_POLICY.UPDATED
- ACTION.DELETED
- AGREEMENT.DELETED
- AGREEMENT_LANGUAGE.DELETED
- AGREEMENT_LANGUAGE_REVISION.DELETED
- APPLICATION.DELETED

- AUTHORIZE_POLICY.DELETED
- CERTIFICATE.DELETED
- DEVICE.DELETED
- DEVICE_AUTHENTICATION_POLICY.DELETED
- FIDO_POLICY.DELETED
- FLOW.DELETED
- FLOW_DEFINITION.DELETED
- GROUP.DELETED
- IDENTITY_PROVIDER.DELETED
- IDP_ATTRIBUTE.DELETED
- INSTANT_MESSAGING_DELIVERY_SETTINGS.DELETED
- KEY.DELETED
- LICENSE.DELETED
- NOTIFICATION_POLICY.DELETED
- ORGANIZATION.DELETED
- POLICY.DELETED
- RISK_POLICY_SET.DELETED
- SAML_ATTRIBUTE.DELETED
- SCHEMA_ATTRIBUTE.DELETED
- SIGN_ON_POLICY_ASSIGNMENT.DELETED
- VERIFY_POLICY.DELETED
- DEVICE.UNBLOCKED
- DEVICE.BLOCKED
- NOTIFICATION.REJECTED
- DEVICE.ACTIVATED
- DEVICE.LOCKED
- DEVICE.UNLOCKED
- ROLE.CREATED
- ROLE.UPDATED
- ROLE.DELETED

CloudWatch pipelines configuration for PingIdentity PingOne

Collects audit logs from PingIdentity PingOne using OAuth2 authentication.

Configure the PingIdentity PingOne source with the following parameters:

```
source:
  pingidentity_pingone:
    range: "P7D"
    region: "NA"
    environment_id: "<your-environment-id>"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`range` (optional)

For pulling historical logs. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for the last 21 hours). The default is 0 hours, and the maximum is 90 days.

`region` (optional)

PingOne region code (NA, EU, AP, AU, CA, SG). The default is "NA".

`environment_id` (required)

The PingOne environment ID.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for PingIdentity PingOne API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for PingIdentity PingOne API authentication.

Proofpoint TAP integration configuration

Proofpoint TAP is an advanced email security solution that detects, analyzes, and blocks advanced threats before they reach users' inboxes. CloudWatch pipelines use the Proofpoint TAP SIEM API, Campaign API, and Forensics API to retrieve security event data including message and click events, threat campaign information, and sandbox analysis results.

Topics

- Source configuration for Proofpoint TAP
- CloudWatch pipelines configuration for Proofpoint TAP

Source configuration for Proofpoint TAP

Integrating with Proofpoint TAP

CloudWatch Pipelines use the Proofpoint TAP SIEM API, Campaign API, and Forensics API to retrieve security event data. The SIEM API returns message and click events within configurable time windows, while the Campaign and Forensics APIs provide detailed threat campaign information and sandbox analysis results. All API endpoints are available at `https://tap-api-v2.proofpoint.com`.

To integrate CloudWatch Pipelines with Proofpoint TAP, complete the following steps:

1. Configure Proofpoint TAP authentication credentials:
   - Log in to the Proofpoint TAP Threat Insight Dashboard.
   - Navigate to Settings > Connected Applications > Service Credentials.
   - Generate a new service credential.
   - Note the Service Principal and Secret values.
2. Store the `servicePrincipal` and `secret` in AWS Secrets Manager.
3. Create a CloudWatch Logs log group for the pipeline output.
4. Create an IAM role with AWS Secrets Manager access and a trust relationship for the pipeline.
5. Configure and create the CloudWatch pipeline.

Prerequisites

Before you begin, make sure you have the following:

- An active Proofpoint TAP account with Threat Insight Dashboard access
- A TAP service credential with SIEM, Campaign, and Forensics API access
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager

Authenticating with Proofpoint TAP

Proofpoint TAP uses HTTP Basic authentication with a service principal and secret. The pipeline sends these credentials with each API request to authenticate access to the SIEM, Campaign, and Forensics endpoints.

Configure Authentication for Proofpoint TAP

To configure authentication credentials for the pipeline:

1. Log in to the Proofpoint TAP Threat Insight Dashboard.
2. Navigate to Settings > Connected Applications > Service Credentials.
3. Generate a new service credential.
4. Note the Service Principal and Secret values.
5. Store the `servicePrincipal` and `secret` in AWS Secrets Manager.

Configuring the CloudWatch Pipeline

To configure the pipeline, choose Proofpoint TAP as the data source. Provide the required `service_principal` and `secret` credentials. After you create and activate the pipeline, security event data from Proofpoint TAP will begin flowing into the selected CloudWatch Logs log group.

The following optional parameters are available:

- **Polling intervals** – `siem_polling_interval` (default `PT10M`), `clicks_permitted_polling_interval` (default `PT5M`), `campaign_polling_interval` (default `P1D`)
- **Backfill durations** – `siem_backfill` (default `P1D`), `campaign_backfill` (default `P1D`). Valid range: `PT30S` to `P1D`.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports the following OCSF event classes:

- **Email Activity [4009]** – Message Delivered, Click Permitted
- **Detection Finding [2004]** – Message Blocked, Click Blocked
- **OSINT Inventory Info [5021]** – Campaign, Forensic Evidence

CloudWatch pipelines configuration for Proofpoint TAP

Collects email security event data from Proofpoint TAP using HTTP Basic authentication.

Configure the Proofpoint TAP source with the following parameters:

```
source:
  proofpoint_tap:
    authentication:
      basic:
        service_principal: "${{aws_secrets:proofpoint-tap-account-
credentials:servicePrincipal}}"
        secret: "${{aws_secrets:proofpoint-tap-account-credentials:secret}}"
    range: "P1D"
```

Parameters

`authentication.basic.service_principal` (required)

The Proofpoint TAP service principal for HTTP Basic authentication, stored in AWS Secrets Manager.

`authentication.basic.secret` (required)

The Proofpoint TAP secret for HTTP Basic authentication, stored in AWS Secrets Manager.

`range` (optional)

The historical time period for backfilling data. Uses ISO 8601 duration format. Minimum is `PT30S`, maximum is `P1D`. Default is `P1D`.

Qualys CSAM integration configuration

Qualys CyberSecurity Asset Management (CSAM) provides visibility into your IT asset inventory, software components, and domain intelligence. It discovers and catalogs assets across on-premises, cloud, and container environments, and it tracks the software installed on them, including lifecycle and authorization status. It also identifies vulnerabilities in your internet-facing assets through External Attack Surface Management (EASM), and it detects unresolved and typosquatted domains.

CloudWatch pipelines use the Qualys CSAM REST API to retrieve asset inventory, software components, vulnerability findings, and domain intelligence data from your Qualys tenant. The API provides paginated REST endpoints with JSON Web Token (JWT) bearer token authentication for retrieving security asset data for monitoring and analysis.

Topics

- Source configuration for Qualys CSAM
- CloudWatch pipelines configuration for Qualys CSAM
- Known Qualys CSAM platform limitations
- Troubleshooting the Qualys CSAM integration

Source configuration for Qualys CSAM

Supported API and platform versions

The following table lists the supported API and platform versions.

Component	Version	Notes
Qualys CSAM API	v2.0	CSAM and Global AssetView (GAV) APIs
OCSF schema	v1.5.0	Open Cybersecurity Schema Framework mapping
OAuth 2.0 token exchange	Not applicable	Form-urlencoded username and password exchange for a JWT bearer token

Qualys Cloud Platform prerequisites

Before you begin, make sure you have the following:

- An active Qualys subscription with the Qualys CSAM module enabled
- A Qualys user account with API access permissions to CSAM endpoints
- Your platform-specific Qualys API Gateway hostname, such as `gateway.qg3.apps.qualys.com`

Configure Qualys CSAM

Use the following procedure to create a Qualys Reader user for the integration.

1. Log in to the Qualys Cloud Platform and identify your Qualys API Gateway hostname.
2. From the module picker, choose **Administration** under **Platform And Sensor Management**.

3. On the Administration page, choose the **Users** tab. From the **Create User** menu, choose **Create Reader User**.
4. On the **General Information** tab, complete all required fields: first name, last name, address, country, state, ZIP code, and email address.
5. Choose **User Role** in the left navigation pane, and then configure the following settings:
   - For **User Role**, choose **Reader**.
   - Under **Allow access to**, select **API**. API access is required for the account to authenticate to the Qualys API.
   - Leave **Business Unit** set to **Unassigned** unless your organization uses business units to segment access.
6. Choose **Asset Groups** in the left navigation pane. Use **Add asset groups** to select the asset groups that the user, and therefore the connector, can access. Select all asset groups to retrieve every asset in the subscription, or select specific groups to limit the scope.
7. Choose **Permissions** in the left navigation pane. The extended permissions to manage the VM module, purge host information or history, and manage the PC module are not required for API read access. Leave them cleared unless your process requires this account to manage those modules.
8. Complete the remaining **Options** and **Security** tabs. Configure options such as IP-based login restrictions only if your organization requires them.
9. Choose **Save** or **Create User**. Qualys sends the new user a temporary password.
10. Log in to the Qualys user interface with the temporary password and complete the required password reset. The credentials do not work with the connector until this first-time password reset is complete.
11. Save the final `username` and `password` for use in the AWS connector configuration.

Qualys API Gateway hostnames

The following list shows the Qualys API Gateway hostname for each platform.

US Platform 1

```
gateway.qg1.apps.qualys.com
```

US Platform 2

```
gateway.qg2.apps.qualys.com
```

US Platform 3

```
gateway.qg3.apps.qualys.com
```

US Platform 4

```
gateway.qg4.apps.qualys.com
```

EU Platform 1

```
gateway.qg1.apps.qualys.eu
```

EU Platform 2

```
gateway.qg2.apps.qualys.eu
```

India Platform

```
gateway.qg1.apps.qualys.in
```

Canada Platform

```
gateway.qg1.apps.qualys.ca
```

UAE Platform

```
gateway.qg1.apps.qualys.ae
```

Australia Platform

```
gateway.qg1.apps.qualys.com.au
```

AWS prerequisites

Before you configure the data source, make sure you have the following:

- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called “API caller permissions”.
- An AWS account with permissions to create and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called “Third-party sources (API Pull)”.
- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called “CloudWatch Logs API operations and required permissions for actions”.
- The Qualys username and password from the product configuration procedure

Configure the CloudWatch pipeline resources

1. Store the Qualys credentials in AWS Secrets Manager.
   - Open the AWS Secrets Manager console and choose **Store a new secret**.
   - For the secret type, choose **Other type of secret**.
   - Add a `username` key with your Qualys platform username.
   - Add a `password` key with your Qualys platform password.
   - Name the secret, for example, `qualys-csam-credentials`.
2. Create the destination CloudWatch Logs log group. Use a descriptive name, such as `/aws/cloudwatch/pipelines/qualys-csam`.
3. Create the source IAM role with the required permissions. For information about the required trust and permissions policies, see the section called “CloudWatch pipelines IAM policies and permissions”.
4. Open the CloudWatch console and navigate to **Pipelines**. Choose **Qualys CSAM** as the data source, configure authentication with the Qualys API Gateway hostname, username, and password, and then select the destination log group. For the complete pipeline configuration and parameter reference, see the section called “Pipeline configuration”.
5. Configure the CloudWatch Logs resource policy. The required action depends on how you create the pipeline and whether the log group already exists.
   - When you use the console with a new log group, the resource policy is created automatically.
   - When you use the console with an existing log group that already has a resource policy, the console displays the message *Resource Policy Detected: The log group you selected already has a resource policy. Please verify that this policy includes permissions for pipelines to write to the destination log group. You may need to modify the existing resource policy to add the necessary permissions.* Review and update the existing policy to include the required pipeline write permissions. Add the permissions within five minutes of pipeline creation.
   - When you use the AWS CLI or API, a resource policy is never created automatically. Create the CloudWatch Logs resource policy manually before the pipeline becomes active.
6. Wait a few minutes for the initial data retrieval. Check the destination log group for incoming events and monitor pipeline metrics for successful ingestion.

Authentication mechanism

Qualys CSAM uses OAuth 2.0 token exchange for authentication.

The pipeline sends a form-urlencoded POST request containing the username, password, and `token=true` to the Qualys `/auth` endpoint. The endpoint returns a JSON Web Token (JWT) that the connector sends as a bearer token with subsequent API calls.

The token remains valid for four hours, and the connector automatically renews it before expiration. The connector retrieves credentials securely from AWS Secrets Manager at runtime.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0 and maps Qualys CSAM events to the following OCSF classes.

Event type	Application or endpoint ID	OCSF class	Description
Assets	`POST /rest/2.0/search/am/asset`	Device Inventory Info (5001)	Discovered hosts and devices with operating system, hardware, network interface, agent, and risk score information
Software Components	`POST /rest/2.0/am/asset/component`	Software Inventory Info (5020)	Software packages discovered on assets, including name, version, and installation metadata
Vulnerabilities	`POST /rest/2.0/search/am/easm/vulns`	Vulnerability Finding (2002)	Vulnerabilities detected on external-facing assets, including CVE, CVSS, and Qualys Vulnerability Score (QVS) information
Unresolved Domains	`POST /rest/2.0/am/domain/list`	OSINT Inventory Info (5021)	Unresolved subdomains under monitored domains
Typosquatted Domains	`POST /rest/2.0/am/domain/list?do`	OSINT Inventory Info (5021)	Typosquatted domains discovered through permutation analysis that impersonate legitimate domains

Event type	Application or endpoint ID	OCSF class	Description
	`mainType=TYPOSQUATTED_DOMAINS`		

Events that do not match an OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

CloudWatch pipelines configuration for Qualys CSAM

With this pipeline, you can collect asset inventory, software component, External Attack Surface Management (EASM) vulnerability finding, and domain intelligence data from Qualys CSAM. The pipeline uses the Global AssetView (GAV) and CSAM REST APIs with OAuth 2.0 authentication and a JWT exchange.

The following example shows the complete pipeline configuration:

```
extension:
  aws:
    secrets:
      qualys_csam:
        secret_id: "arn:aws:secretsmanager:<region>:<account-id>:secret:<secret-name>"
        region: "<region>"
        sts_role_arn: "arn:aws:iam::<account-id>:role/<role-name>"
pipeline:
  source:
    qualys_csam:
      hostname: "gateway.qg3.apps.qualys.com"
      range: "P1D"
      authentication:
        oauth2:
          username: "${{aws_secrets:qualys_csam:username}}"
          password: "${{aws_secrets:qualys_csam:password}}"
  sink:
    - cloudwatch_logs:
        log_group: "<qualys-csam-log-group>"
  processor:
    - ocsf:
```

```
        schema:
          qualys_csam: null
        version: "1.5"
        mapping_version: 1.5.0
```

AWS extension parameters

The AWS extension configures AWS Secrets Manager access for securely retrieving credentials at runtime. Define a named secret block, such as `qualys_csam`. Set `secret_id` to the ARN of your secret, `region` to the AWS Region where the secret is stored, and `sts_role_arn` to the IAM role ARN to assume when accessing the secret. Reference credentials in the pipeline with `${{aws_secrets:<name>:<key>}}`. For more information, see the AWS Secrets Manager extension.

Source parameters

Parameters

`hostname` (string, required)

The customer-specific Qualys API Gateway hostname for your platform subscription, such as `gateway.qg3.apps.qualys.com`. Do not include `https://`.

`authentication.oauth2.username` (string, required)

The Qualys platform username for JWT token exchange. Store the value in AWS Secrets Manager and reference it by using `${{aws_secrets:qualys_csam:username}}`.

`authentication.oauth2.password` (string, required)

The Qualys platform password for JWT token exchange. Store the value in AWS Secrets Manager and reference it by using `${{aws_secrets:qualys_csam:password}}`.

`range` (string, optional)

The lookback duration for the initial data backfill, in ISO 8601 duration format. The minimum is one hour (`PT1H`), the maximum is one day (`P1D`), and the default is `P1D`.

> **Note**
>
> The connector retrieves credential values from AWS Secrets Manager. You can find the Qualys user name in the Qualys Cloud Platform on the **Administration** page, on the **Users**

tab. You can use the same credentials as Qualys VMDR or Policy Compliance (PC) if the account has access to the CSAM module.

Sink parameters

The sink delivers processed events to a CloudWatch Logs log group. Set `log_group` to the name of your destination log group, such as `qualys-csam-log-group`. For complete configuration options, see the CloudWatch Logs sink.

Processor parameters

The processor transforms ingested events into OCSF format using the `qualys_csam` schema key. Set `schema.qualys_csam` to `null`; the presence of the key activates the Qualys CSAM schema mapping. Set `version` to `"1.5"` and `mapping_version` to `1.5.0`. For the full parameter reference, see the OCSF processor.

Known Qualys CSAM platform limitations

The following table describes known Qualys CSAM platform limitations.

Limitation	Details	Impact
Customer-specific API Gateway hostname	The API Gateway URL varies by Qualys platform region. There is no universal endpoint.	You must identify your platform and configure the correct gateway hostname.
API rate limits	With the default Standard API Service, Qualys permits 300 API calls per subscription for each API in a rolling 3,600-second window. When the limit is exceeded, Qualys returns HTTP 409 Conflict with error code 1965 and an `X-RateLimit-ToWait-Sec` header.	The connector waits for the interval in the `X-RateLimit-ToWait-Sec` header and then retries. Large environments might experience a slower initial backfill.
Pagination on list endpoints	CSAM uses cursor-based pagination for assets, software, vulnerabilities,	Large inventories have higher retrieval latency. The connector handles pagination internally.

Limitation	Details	Impact
	and domains. Each page requires a separate API call.	
JWT token expiration	The bearer token expires after approximately four hours.	The connector automatically renews the token before expiration. No action is required.
No sort-order support	The CSAM API does not document an ascending or descending sort parameter.	Records are not guaranteed to arrive in a specific order within a window.
Shared credentials with VMDR	Qualys CSAM and VMDR share platform credentials but use different API endpoints and hostnames.	If you use both connectors, you can store credentials separately for each connector or use a shared secret.
CSAM module licensing	Qualys CSAM is separately licensed. Individual EASM vulnerability and domain APIs require their corresponding submodules.	API calls fail if your subscription does not include the corresponding module.

Troubleshooting the Qualys CSAM integration

Common errors

The following table lists common errors and their resolutions.

Error	Likely cause	Resolution or connector behavior
401 Unauthorized	Invalid or expired credentials	The connector automatically renews credentials and retries up to six times. Verify the username and password in AWS Secrets Manager.
403 Forbidden	Insufficient permissions or the CSAM module is not licensed	Make sure that API access is enabled for the Qualys account and that the subscription

Error	Likely cause	Resolution or connector behavior
		includes the CSAM module. This error is not retryable and stops processing immediately.
409 Conflict (CODE 1965)	API rate limit exceeded	The connector reads the `X-RateLimit-ToWait-Sec` header and retries after the specified wait time.
409 Conflict (CODE 1960)	Concurrent call limit exceeded	The connector retries with progressive backoff of 5, 10, 30, 60, 120, and 300 seconds.
5xx Server Error	Qualys platform server-side issue	The connector automatically retries with backoff intervals of 1, 2, 5, 10, 20, and 40 seconds.
No data appears in the log group	The pipeline resource policy was not added within five minutes	Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.
Empty results	No scans or discovery ran on the hosts in the configured time range	Verify that scans are active and that hosts were evaluated within the range window.

Identify your Qualys API Gateway hostname

- The CSAM connector uses the Qualys API Gateway hostname, such as `gateway.qg3.apps.qualys.com`. This differs from the VMDR API hostname, such as `qualysapi.qg3.apps.qualys.com`.
- Log in to the Qualys Cloud Platform and navigate to **Help** > **About** to identify your platform assignment.
- If you are not sure which platform you use, see Qualys Platform Identification.
- Enter only the domain. Don't include `https://` in the hostname.

Difference between CSAM and VMDR hostnames

- Qualys VMDR uses `qualysapi.<platform>.apps.qualys.com`, such as `qualysapi.qg3.apps.qualys.com`.
- Qualys CSAM uses `gateway.<platform>.apps.qualys.com`, such as `gateway.qg3.apps.qualys.com`.
- Both hostnames correspond to the same Qualys platform but use different API surfaces. Use the `gateway` hostname for the Qualys CSAM connector.

Qualys Policy Audit integration configuration

Qualys Policy Audit (PA) is a cloud-native configuration assessment and compliance monitoring solution. It evaluates IT assets against security benchmarks, such as CIS, DISA STIG, and vendor hardening guides, and regulatory frameworks, such as PCI-DSS, HIPAA, SOX, and NIST.

As part of the Qualys Cloud Platform, it uses lightweight Cloud Agents and Scanner Appliances to assess on-premises, cloud, and hybrid environments. It includes out-of-the-box compliance policies, continuous assessment scheduling, and evidence-based reporting.

Compliance posture findings report whether each host passed or failed a policy control. CloudWatch pipelines use the Policy Compliance Reporting Service (PCRS) streaming API to ingest these findings. The connector uses the following retrieval flow:

1. Authenticate through OAuth 2.0 by exchanging credentials for a JSON Web Token (JWT).
2. Resolve all host IDs with their `policyId` and `subscriptionId`.
3. Retrieve per-host compliance posture findings within time-partitioned windows.

The following table lists the supported API and platform versions.

Component	Version	Notes
Qualys PCRS API	v5.0	Policy Compliance Reporting Service posture endpoints under `/pcrs/5.0/posture/*`
Qualys Cloud Platform	Platform-specific	Regional gateways such as qg1, qg2, and qg3. See Qualys Platform Identification.

Component	Version	Notes
OCSF schema	v1.5.0	Open Cybersecurity Schema Framework mapping
OAuth 2.0	2	Form-urlencoded username and password JWT token exchange

Topics

- Source configuration for Qualys Policy Audit
- CloudWatch pipelines configuration for Qualys Policy Audit

Source configuration for Qualys Policy Audit

Prerequisites

Before you begin, make sure you have the following:

- An active Qualys subscription with the Policy Compliance (Policy Audit) module enabled
- A Qualys user account with API access permissions to PCRS endpoints
- A Qualys API user account that is not SSO-only. API authentication uses a username and password, so the account must be able to authenticate directly.
- Your platform-specific Qualys API Gateway hostname, such as `gateway.qg3.apps.qualys.com`
- The final Qualys username and password from the Qualys console procedure, after the first-time password reset is complete
- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called “API caller permissions”.
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called “Third-party sources (API Pull)”.
- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called “CloudWatch Logs API operations and required permissions for actions”.

Integrating with Qualys Policy Audit

To integrate Qualys Policy Audit with CloudWatch pipelines, complete the following high-level steps:

- Identify your platform-specific Qualys API Gateway hostname.
- Create a Qualys Reader user with API access and the required asset groups.
- Store the Qualys username and password in AWS Secrets Manager.
- Create a destination CloudWatch Logs log group and the source IAM role.
- Create a CloudWatch pipeline with Qualys Policy Audit as the data source.
- Verify that compliance posture findings are flowing into the configured log group.

Configure Qualys Policy Audit

1. **Identify your Qualys API Gateway hostname**

   Log in to the Qualys Cloud Platform. Use the Qualys Platform Identification page to determine your platform-specific API Gateway hostname, such as `gateway.qg3.apps.qualys.com`.

2. **Open the Administration module**

   From the module picker, select **Administration** under Platform And Sensor Management.

3. **Start the New User wizard**

   On the Administration page, choose the **Users** tab. Choose **Create User**, and then choose **Create Reader User**.

4. **Enter General Information**

   On the General Information tab, complete the required fields marked with an asterisk: First Name, Last Name, Address 1, Country, State, ZIP Code, and E-mail Address.

5. **Set the User Role and enable API access**

   Choose **User Role** in the left navigation. Set **User Role** to **Reader**. Under **Allow access to**, select **API**. You must select **API**, because this setting is what allows the account to authenticate to the Qualys API for the connector. Leave **Business Unit** set to **Unassigned** unless your organization uses Business Units to segment access.

6. **Assign Asset Groups**

Choose **Asset Groups** in the left navigation. Use **Add asset groups** to select the asset groups that should be visible to this user and to the connector. Select **All** to retrieve data for every asset in the subscription, or select specific groups to limit the scope.

7. **Review Permissions**

   Choose **Permissions** in the left navigation. The Extended Permissions options for Manage VM module, Purge host information/history, and Manage PC module grant administrative rights and are not required for API read access. Leave them unselected unless your process requires the account to manage those modules.

8. **Complete the remaining tabs and create the user**

   Continue through the remaining Options and Security tabs, changing settings only if your organization has specific requirements such as IP-based login restrictions. Create the user. Qualys emails the new user a temporary password.

   Log in once through the Qualys user interface with the temporary password and complete the required password reset. The API user credentials do not work for the connector until this first-time password reset is complete. Save the final username and password for the pipeline configuration.

Authenticating with Qualys Policy Audit

Qualys Policy Audit uses OAuth 2.0 with a username and password exchange for a JWT. The connector performs the following authentication flow:

1. Retrieves the credentials securely from AWS Secrets Manager at runtime.
2. Sends a form-urlencoded `POST` request to the Qualys `/auth` endpoint with `username`, `password`, and `token=true`.
3. Receives the JWT as a raw plain-text string. The token remains valid for approximately four hours.
4. Uses `Authorization: Bearer <token>` for all subsequent API requests and automatically renews the token before it expires.

Configure the CloudWatch pipeline resources

1. **Store credentials in AWS Secrets Manager**

In AWS Secrets Manager, choose **Store a new secret**, and then choose **Other type of secret**. Add the following key-value pairs:

- `username` – Your Qualys platform username
- `password` – Your Qualys platform password

Name the secret, for example `qualys-policyaudit/pipeline-credentials`.

2. **Create the destination CloudWatch Logs log group**

   In the CloudWatch console, choose **Logs**, **Log groups**, and **Create log group**. Choose a descriptive name, such as `/aws/cloudwatch/pipelines/qualys-policyaudit`.

3. **Create the source IAM role**

   Create an IAM role with the required permissions. For information about the required policies and permissions, see the section called “CloudWatch pipelines IAM policies and permissions”.

4. **Create the CloudWatch pipeline**

   In the CloudWatch console, navigate to Pipelines and choose **Qualys Policy Audit** as the data source. Configure authentication with the Qualys API Gateway hostname, username, and password. Select the destination log group. For the complete pipeline YAML configuration and parameter reference, see the section called “Pipeline configuration”.

5. **Configure the CloudWatch Logs resource policy**

   The behavior depends on whether the destination log group is new or already exists. For a new log group created through the console, the resource policy is created automatically and no action is required. If the selected log group already has a resource policy, the console displays a Resource Policy Detected warning. In that case, verify that the existing policy includes permissions for pipelines to write to the destination log group, and add or update the CloudWatch Logs resource policy within five minutes of pipeline creation.

   > **Note**
   >
   > When you create the pipeline by using the AWS CLI or API, a resource policy is never created automatically, regardless of whether the log group is new or existing. You must create the resource policy manually before the pipeline becomes active.

6. **Verify data flow**

Allow time for the initial data retrieval to complete. Check the destination log group for incoming events, and monitor pipeline metrics for successful ingestion.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0. Qualys Policy Audit compliance posture findings are mapped to the Compliance Finding OCSF class, capturing per-host, per-control evaluation results including pass, fail, and error status; control statements; rationale; remediation guidance; severity; and scan timestamps.

Compliance Finding (2003)

The Compliance Posture event from the `/pcrs/5.0/posture/postureInfo` endpoint maps to Compliance Finding (2003). Each event contains a per-host, per-control compliance evaluation including pass, fail, or error status; control statements; rationale; remediation; severity or criticality; and scan timestamps.

> **Note**
>
> Events that do not match an OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

Known platform limitations

The following table describes known Qualys Policy Audit platform limitations.

Limitation	Details	Impact
Backfill window	The first-run backfill is bounded by `range`, with a minimum of `P1D`, maximum of `P5D`, and default of `P5D`.	Historical posture data older than the backfill window is not retrieved on the first run.
Posture availability delay	Posture data becomes available only after a compliance scan completes and is evaluated by the Qualys	Findings are delivered only after the Qualys platform finishes evaluating a scan, so delivery lags scan completion.

Limitation	Details	Impact
	platform. Qualys does not document an exact delay.	
No sort-order support	The PCRS API does not document an ascending or descending sort parameter.	Records are not guaranteed to arrive in a specific order within a window.
Shared API rate limit	With the default Standard API Service, Qualys permits 300 API calls per subscription for each API in a rolling 3,600-second window. When the limit is exceeded, Qualys returns HTTP 409 Conflict with error code 1965 and an `X-RateLimit-ToWait-Sec` header. Qualys Support can customize the limit for your subscription.	The pipeline shares one quota with every other user and integration in the subscription, so heavy API use elsewhere slows pipeline ingestion, especially during the initial backfill. The connector waits for the interval in the header and retries, so throughput is reduced but data is not lost.
Concurrent call limit	With the default Standard API Service, Qualys permits two concurrent API calls for each API. Qualys checks the concurrency limit before the rate limit and returns HTTP 409 Conflict with error code 1960 when the limit is exceeded. Concurrency errors take precedence over rate-limit errors and don't include rate-limit headers.	High-volume environments might experience throttling during the initial backfill. The connector retries with backoff, so throughput is reduced but data is not lost.
Customer-specific hostname	There is no universal endpoint. The API Gateway hostname is assigned per Qualys subscription and region.	You must supply the exact gateway fully qualified domain name for your platform.
Token time to live	JWT tokens expire approximately four hours after issue.	The connector automatically renews the token before it expires.

Troubleshooting

The following table lists common errors and their resolutions.

Error	Likely cause	Resolution or connector behavior
401 Unauthorized	Invalid or expired credentials, or credentials rotated in AWS Secrets Manager	The connector renews the token and retries up to six times. Verify that the username and password in AWS Secrets Manager are current, API access is enabled, and the account is not SSO-only.
403 Forbidden	Insufficient permissions for the API user	This error is not retryable, and the connector stops immediately. Verify that the Reader user has API access and the required asset groups assigned.
5xx Server Error	Transient Qualys platform error	The connector retries with exponential backoff at 1, 2, 5, 10, 20, and 40 seconds.
No data in the log group	The pipeline resource policy was not added within five minutes of creation.	Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.
Wrong hostname or connection failure	A VMDR hostname was used instead of the PCRS gateway hostname, or the hostname includes a scheme.	Use the `gateway.<platform>.apps.qualys.com` form without `https://`.

Identify the Qualys API Gateway hostname

- The Policy Audit connector uses the Qualys API Gateway hostname, such as `gateway.qg3.apps.qualys.com`. This differs from the VMDR API hostname, such as `qualysapi.qg3.apps.qualys.com`.

- Identify your platform from the platform identifier embedded in your Qualys username. For example, the character 3 indicates US Platform 3, a hyphen indicates EU Platform 1, and the character C indicates the US federal platform.
- Use the Qualys Platform Identification page to map your platform identifier to the gateway hostname.
- Provide only the domain. Do not include `https://`.

Policy Audit and VMDR use different hostnames

- Qualys VMDR uses `qualysapi.<platform>.apps.qualys.com` on most platforms, such as `qualysapi.qg3.apps.qualys.com`. US Platform 1 and EU Platform 1 are exceptions and use the shorter legacy forms `qualysapi.qualys.com` and `qualysapi.qualys.eu`, with no platform segment and no `apps` segment.
- Qualys Policy Audit uses `gateway.<platform>.apps.qualys.com`, such as `gateway.qg3.apps.qualys.com`.
- Both hostnames are on the same Qualys platform but use different API surfaces. Use the gateway hostname for this connector.

No compliance findings are returned

- Confirm that the Policy Compliance module is active in your Qualys subscription.
- Verify that compliance policies are assigned to hosts or asset groups and that scans have completed.
- Check the `range` parameter. The default is `P5D`. If no scans occurred in that window, no data is returned.

References

- the section called “CloudWatch pipelines IAM policies and permissions”
- the section called “Monitoring pipelines”
- the section called “Troubleshooting”
- Qualys API (VM, PA) User Guide
- Qualys Policy Compliance Guide
- Qualys API Limits

- Qualys Platform Identification
- OCSF schema v1.5.0

CloudWatch pipelines configuration for Qualys Policy Audit

Collects compliance posture findings from Qualys Policy Audit by using the PCRS streaming API with OAuth 2.0 authentication and a JWT exchange.

Configure the Qualys Policy Audit pipeline with the following YAML:

```
extension:
  aws:
    secrets:
      qualys-policyaudit-credentials:
        secret_id: "<<secret-arn>>"
        region: "<<secret-region>>"
        sts_role_arn: "<<secret-access-role-arn>>"

pipeline:
  source:
    qualys_policyaudit:
      hostname: "gateway.qg3.apps.qualys.com"
      authentication:
        oauth2:
          username: "${{aws_secrets:qualys-policyaudit-credentials:username}}"
          password: "${{aws_secrets:qualys-policyaudit-credentials:password}}"
      range: "P5D"               # optional (default); from P1D to P5D

  processor:
    - ocsf:
        schema:
          qualys_policyaudit: null
        version: "1.5"
        mapping_version: 1.5.0

  sink:
    - cloudwatch_logs:
        log_group: "/aws/cloudwatch/pipelines/qualys-policyaudit"
```

Source parameters

`hostname` (required)

The Qualys API Gateway hostname, such as `gateway.qg3.apps.qualys.com`. Do not include a scheme. The connector adds `https://` automatically. The hostname varies by platform, such as qg1, qg2, or qg3.

`authentication.oauth2.username` (required)

The Qualys platform username for the JWT token exchange. Store it in AWS Secrets Manager and reference it with `${{aws_secrets:qualys-policyaudit-credentials:username}}`.

`authentication.oauth2.password` (required)

The Qualys platform password for the JWT token exchange. Store it in AWS Secrets Manager and reference it with `${{aws_secrets:qualys-policyaudit-credentials:password}}`.

`range` (optional)

How far back to retrieve posture data on the first run. Use an ISO 8601 duration from `P1D` through `P5D`. Default: `P5D`.

AWS Secrets Manager extension parameters

Define a named secret block, such as `qualys-policyaudit-credentials`, under `extension.aws.secrets`. Set `secret_id` to the ARN of the AWS Secrets Manager secret, `region` to the AWS Region where the secret is stored, and `sts_role_arn` to the role that accesses the secret. Reference credentials with `${{aws_secrets:<name>:<key>}}`.

Processor parameters

The `ocsf` processor transforms ingested events into OCSF format. Under `schema`, specify the single source key `qualys_policyaudit` with a null value. Set `version` to `1.5` and `mapping_version` to `1.5.0`.

Sink parameters

The `cloudwatch_logs` sink delivers processed events to a CloudWatch Logs log group. Set `log_group` to the name of your destination log group, such as `/aws/cloudwatch/pipelines/qualys-policyaudit`. The sink writes to the AWS Region in which you create the pipeline, and you do not specify an AWS Region in the sink configuration.

> **Note**
>
> The credential values are retrieved from AWS Secrets Manager. You can find your Qualys credentials in the Qualys Cloud Platform under Administration > User Management. The same credentials used for Qualys VMDR can be used for this connector if the account has access to the Policy Compliance module and can authenticate with a username and password.

Qualys VMDR integration configuration

Qualys Vulnerability Management, Detection and Response (VMDR) is a cloud-native vulnerability management platform. It provides continuous discovery, assessment, prioritization, and remediation of vulnerabilities across on-premises, cloud, container, and hybrid environments. It combines asset discovery and vulnerability scanning with prioritization based on TruRisk, the proprietary Qualys risk-scoring model, and integrated patch management.

CloudWatch pipelines poll the Qualys REST APIs to ingest vulnerability knowledge base data, asset inventory, host detection findings, and platform activity logs.

Topics

- Source configuration for Qualys VMDR
- CloudWatch pipelines configuration for Qualys VMDR

Source configuration for Qualys VMDR

Integrating with Qualys VMDR

CloudWatch pipelines use versions 2.0, 4.0, and 5.0 of the Qualys Vulnerability Management (VM) and Policy Compliance (PC) APIs across the supported endpoints. The APIs use HTTP Basic authentication and return XML for Knowledge Base, Assets, and Host Detection data, and CSV for Activity Log data. The connector handles the response parsing internally.

To integrate CloudWatch pipelines with Qualys VMDR, complete the following high-level steps:

- Identify the API hostname for your Qualys platform.
- Create a Qualys user with the Manager role, enable API access, and assign the asset groups that the connector can access.

- Store the Qualys username and password in AWS Secrets Manager.
- Create a CloudWatch pipeline with Qualys VMDR as the data source.
- Verify that data is flowing into the destination CloudWatch Logs log group.

Supported API and platform versions

The following table lists the supported API and platform versions.

Component	Version	Notes
Qualys Vulnerability Management and Policy Compliance (VMPC) API	v2.0, v4.0, v5.0	Multiple API versions are used across the supported endpoints
Open Cybersecurity Schema Framework (OCSF) schema	v1.5.0	Event mapping to the OCSF schema
HTTP Basic authentication	Not applicable	Base64-encoded `username:password` credentials with each request

Prerequisites

Before you begin, make sure you have the following:

- An active Qualys VMDR subscription with API access enabled
- A Qualys user account with the Manager role, or a role with equivalent permissions, and with API access enabled
- The hostname for your Qualys platform, such as `qualysapi.qg3.apps.qualys.com`
- A Qualys subscription that includes the VMDR module
- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called “API caller permissions”.
- An AWS account with permissions to create and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called “Third-party sources (API Pull)”.

- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called “CloudWatch Logs API operations and required permissions for actions”.

Configure Qualys VMDR

To configure a Qualys user for the integration:

1. Log in to the Qualys Cloud Platform and identify your Qualys API hostname. You can find your platform assignment under **Help > About** or use the Qualys Platform Identification page.
2. From the module picker, choose **Administration** under Platform and Sensor Management.
3. On the Administration page, choose the **Users** tab, and then choose **Create User**.
4. On the **General Information** tab, complete the required fields: First Name, Last Name, Address 1, Country, State, ZIP Code, and E-mail Address.
5. Choose **User Role** in the left navigation pane. Set the role to **Manager**, and under **Allow access to**, select **API**. Leave Business Unit set to Unassigned unless your organization uses business units to segment access.
6. Choose **Asset Groups**. Add the asset groups that should be visible to this user, and therefore to the connector. Select all asset groups if the connector should retrieve data for every asset in the subscription, or select specific groups to limit the scope.
7. Choose **Permissions**. The extended permissions to manage the VM module, purge host information or history, and manage the PC module are not required for API read access. Leave them cleared unless your process requires this account to manage those modules.
8. Complete the remaining Options and Security tabs. Configure settings such as IP-based login restrictions only if required by your organization, and then create the user. Qualys emails the new user a temporary password.
9. Log in to the Qualys user interface with the temporary password and complete the required password reset. The API credentials do not work with the connector until this first-time password reset is complete.
10. Save the final `username` and `password` for the pipeline configuration.

Qualys platform hostnames

Use the hostname that corresponds to your Qualys platform. Do not include `https://`.

US Platform 1

```
qualysapi.qualys.com
```

US Platform 2

```
qualysapi.qg2.apps.qualys.com
```

US Platform 3

```
qualysapi.qg3.apps.qualys.com
```

US Platform 4

```
qualysapi.qg4.apps.qualys.com
```

EU Platform 1

```
qualysapi.qualys.eu
```

EU Platform 2

```
qualysapi.qg2.apps.qualys.eu
```

India Platform

```
qualysapi.qg1.apps.qualys.in
```

Canada Platform

```
qualysapi.qg1.apps.qualys.ca
```

UAE Platform

```
qualysapi.qg1.apps.qualys.ae
```

Australia Platform

```
qualysapi.qg1.apps.qualys.com.au
```

Authenticating with Qualys VMDR

Qualys VMDR uses HTTP Basic authentication. The pipeline sends Base64-encoded `username:password` credentials in an Authorization header with each API request. The credentials are retrieved securely from AWS Secrets Manager at runtime and are not stored in the pipeline configuration.

Configure the CloudWatch pipeline resources

1. In AWS Secrets Manager, choose **Store a new secret**, and then choose **Other type of secret**. Add a `username` key with your Qualys API username and a `password` key with your Qualys API password. Name the secret, for example, `qualys-vmdr/pipeline-credentials`.
2. Create the destination CloudWatch Logs log group, for example, `/aws/cloudwatch/pipelines/qualys-vmdr`.
3. Create the source IAM role with the required permissions. For information about the required policies, see the section called "CloudWatch pipelines IAM policies and permissions".
4. In the CloudWatch console, navigate to Pipelines and choose **Qualys VMDR** as the data source. Configure the Qualys hostname and authentication credentials, and select the destination log group.
5. Configure the CloudWatch Logs resource policy:
   - For a new log group created in the console, the resource policy is created automatically.
   - If the selected log group already has a resource policy, review and update it within 5 minutes of pipeline creation to include the required pipeline write permissions.
   - When you create the pipeline with the AWS CLI or API, a resource policy is not created automatically. Create the policy manually before the pipeline becomes active.
6. Wait a few minutes for the initial data retrieval. Check the destination log group for incoming events and monitor pipeline metrics for successful ingestion.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0. Qualys VMDR events are mapped to OCSF classes based on the data type. Vulnerability definitions map to Vulnerability Finding, asset inventory maps to Device Inventory Info, and activity log events map to Authentication, API Activity, Account Change, or Entity Management based on the action type.

Event type	Application or endpoint ID	OCSF event class	Description
Vulnerability (Knowledge Base)	`/api/4.0/fo/knowledge_base/vuln/?action=list`	Vulnerability Finding (2002)	QID, CVE IDs, severity level 1 through 5, solution text, vendor references, exploitability,

Event type	Application or endpoint ID	OCSF event class	Description
			patchable flag, and published dates
Assets	`/api/5.0/fo/asset/host/?action=list`	Device Inventory Info (5001)	Host asset inventory including IP, operating system, DNS, NetBIOS, agent status, and scan timestamps
Activity Log – login and logout	`/api/2.0/fo/activity_log/?action=list`	Authentication (3002)	User login and logout events with source IP and user role
Activity Log – all non-authentication actions	`/api/2.0/fo/activity_log/?action=list`	API Activity (6003)	API requests, appliance downloads, and vulnerability edits, plus operational actions such as scan launch, scan completion, and report generation. Actions that do not match a more specific class map to API Activity with an activity of Other.
Activity Log – activate, set password, and set authentication	`/api/2.0/fo/activity_log/?action=list`	Account Change (3001)	Account activation, password changes, and credential setup events
Activity Log – administrative configuration, appliance, and asset group	`/api/2.0/fo/activity_log/?action=list`	Entity Management (3004)	Administrative configuration changes, scanner appliance create, update, and delete operations, and asset group and domain management

Ingestion-only event types

The following event type is ingested but is not currently mapped to an OCSF class:

Event type	Application or endpoint ID	Description
Asset Host Detection	`/api/5.0/fo/asset/host/vm/detection/?action=list`	Per-host vulnerability detection records. The records contain nested detection arrays that require sub-array record extraction, which is not currently supported for OCSF mapping.

Activity log actions that do not match a more specific class, such as scan launch, scan completion, and report generation, are mapped to API Activity (6003) with an activity of Other. They are not ingestion-only.

Events that do not match an OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

Known platform limitations

The following table describes known Qualys VMDR platform limitations.

Limitation	Details	Impact
Customer-specific API hostname	The API base URL varies by Qualys platform. There is no universal endpoint.	You must identify your platform and configure the correct hostname.
XML and CSV response formats	The APIs return XML for Knowledge Base, Assets, and Host Detection data, and CSV for Activity Log data. There is no JSON response option.	The connector handles the parsing internally.
API rate limits	With the default Standard API Service, Qualys permits 300 API calls per subscription for each API in a rolling 3,600-second window.	The connector backs off and retries based on the rate-limit headers.

Limitation	Details	Impact
	When the limit is exceeded, Qualys returns HTTP 409 Conflict with error code 1965 and an `X-RateLimit-ToWait-Sec` header.	
Concurrent call limit	With the default Standard API Service, Qualys permits two concurrent API calls per subscription for each API.	High-volume environments might experience throttling during the initial backfill.
No OAuth 2.0 support	The version 2, 4, and 5 API endpoints support only HTTP Basic authentication.	Credentials must be rotated manually. There is no token refresh mechanism.
Asset Host Detection nested arrays	Each host record contains multiple detection entries in a nested array.	These events are ingested without Open Cybersecurity Schema Framework (OCSF) mapping because sub-array extraction is not supported.
Server-driven pagination	Qualys caps the page size at 1,000 records for Assets, 100 records for Host Detection, and 1,000 records for Activity Log.	Large datasets require multiple paginated requests in each polling cycle.
Activity Log shared across modules	The Activity Log captures events from all Qualys modules, not only VMDR.	Some ingested events can relate to Policy Compliance (PC), Web Application Scanning (WAS), or CyberSecurity Asset Management (CSAM).

Troubleshooting

Error	Likely cause	Resolution or connector behavior
401 Unauthorized	Invalid or expired credentials	Verify the username and password stored in AWS Secrets Manager, and make sure that the Qualys account has API access enabled.
403 Forbidden	Insufficient permissions or API access is not enabled	Make sure that the Qualys user has the Manager role, or a role with equivalent permissions, and that the API access option is enabled.
409 Conflict (CODE 1965)	API rate limit exceeded	The connector reads the `X-RateLimit-ToWait-Sec` header and retries after the specified wait time.
409 Conflict (CODE 1960)	Concurrent call limit exceeded	The connector retries with progressive backoff of 5, 10, 30, 60, 120, and 300 seconds.
No data appears in the log group	The pipeline resource policy was not added within 5 minutes	Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.
XML parse errors	Unexpected response format or API version change	Verify that the hostname is correct and that the Qualys subscription includes access to the VMDR module.

During initial backfill, the connector retrieves data for the configured `range` across all four streams. If rate limiting occurs, the connector handles it automatically and no action is required. Enterprise Qualys subscriptions can have higher rate limits; contact Qualys Support to verify your API tier.

The Qualys Activity Log is shared across VMDR, Policy Compliance, Web Application Scanning, and CyberSecurity Asset Management. Events from other modules are ingested and available for search, but might not include VMDR-specific context. This is expected behavior.

CloudWatch pipelines configuration for Qualys VMDR

Collects vulnerability management data, asset inventory, host detection findings, and platform activity logs from Qualys VMDR using HTTP Basic authentication.

The following example shows the complete pipeline configuration:

```
extension:
  aws:
    secrets:
      qualys-vmdr-credentials:
        secret_id: "<secret-arn>"
        region: "<secret-region>"
        sts_role_arn: "<secret-access-role-arn>"
pipeline:
  source:
    qualys_vmdr:
      hostname: "qualysapi.qg3.apps.qualys.com"
      range: "P1D"
      authentication:
        basic:
          username: "${{aws_secrets:qualys-vmdr-credentials:username}}"
          password: "${{aws_secrets:qualys-vmdr-credentials:password}}"
  sink:
    - cloudwatch_logs:
        log_group: "<qualys-vmdr-log-group>"
  processor:
    - ocsf:
        schema:
          qualys_vmdr: null
        version: "1.5"
        mapping_version: 1.5.0
```

The name in each `aws_secrets` reference must match the secret block name under `extension.aws.secrets`.

The following four fields are the only source fields that you configure for `qualys_vmdr`. A configuration that sets any other source field fails validation.

Source parameters

`hostname` (required)

The customer-specific Qualys API hostname for your platform subscription, for example, `qualysapi.qg3.apps.qualys.com`. Do not include `https://`.

`authentication.basic.username` (required)

The Qualys API username. Store the value in AWS Secrets Manager and reference it using `${{aws_secrets:qualys-vmdr-credentials:username}}`.

`authentication.basic.password` (required)

The Qualys API password. Store the value in AWS Secrets Manager and reference it using `${{aws_secrets:qualys-vmdr-credentials:password}}`.

`range` (optional)

The lookback duration for initial data collection, in ISO 8601 duration format. The minimum is one hour (`PT1H`) and the maximum is one day (`P1D`). The value applies to all four streams. If you omit it, each stream backfills `P1D`.

> **Note**
>
> The credential values are retrieved from AWS Secrets Manager. You can manage the Qualys API user in the Qualys Cloud Platform under Administration > User Management.

AWS extension parameters

The AWS extension configures AWS Secrets Manager access so that the pipeline can retrieve credentials at runtime. Define a named secret block, such as `qualys-vmdr-credentials`. Set `secret_id` to the ARN of your secret, `region` to the AWS Region where the secret is stored, and `sts_role_arn` to the IAM role ARN to assume when accessing the secret. Reference the credentials in the pipeline with `${{aws_secrets:<name>:<key>}}`. For more information, see the AWS Secrets Manager extension.

Sink parameters

The sink delivers processed events to a CloudWatch Logs log group. Set `log_group` to the name of your destination log group, such as `/aws/cloudwatch/pipelines/qualys-vmdr`. For complete configuration options, see the CloudWatch Logs sink.

Processor parameters

The processor transforms ingested events into Open Cybersecurity Schema Framework (OCSF) format by using the `qualys_vmdr` schema key. Set `schema.qualys_vmdr` to `null`; the presence of the key activates the Qualys VMDR schema mapping. Set `version` to `"1.5"` and `mapping_version` to `1.5.0`. Events that do not match an OCSF mapping pass through to the sink unchanged. For the full parameter reference, see the OCSF processor.

SailPoint ISC integration configuration

SailPoint Identity Security Cloud (ISC) is a cloud-based identity governance and administration platform. It provides identity security capabilities such as access certifications (periodic reviews of who has access to what), provisioning (granting and removing access), password management, and single sign-on. The platform generates audit events covering authentication, user management, provisioning, access requests, certifications, and system configuration changes. CloudWatch pipelines integrate with SailPoint ISC through its Search API to continuously collect identity governance audit events for centralized security monitoring and analysis.

Topics

- Source configuration for SailPoint ISC
- CloudWatch pipelines configuration for SailPoint ISC

Source configuration for SailPoint ISC

Supported API and schema versions

- **SailPoint ISC Search API v2026** – Primary endpoint for retrieving audit events.
- **OCSF schema v1.5.0** – Open Cybersecurity Schema Framework mapping.
- **OAuth 2.0 Client Credentials** – Token-based authentication using Personal Access Token (PAT) credentials.

Prerequisites

Before you begin, make sure you have the following:

- An active SailPoint Identity Security Cloud tenant with API access enabled.
- A user account with the `ORG_ADMIN` user level, which is required to access all audit events.
- Your SailPoint tenant name and API domain.
- An AWS account with permissions to create and manage CloudWatch pipelines. For more information, see the section called "API caller permissions".
- An AWS account with permissions to create and update secrets in AWS Secrets Manager, and a source role that can retrieve the stored credentials. For more information, see the section called "Third-party sources (API Pull)".
- An AWS account with permissions to create and manage CloudWatch Logs log groups. For more information, see the section called "CloudWatch Logs API operations and required permissions for actions".

Configuring SailPoint ISC

To configure SailPoint ISC for the integration:

1. Sign in to SailPoint Identity Security Cloud as an `ORG_ADMIN` user.
2. In the SailPoint Admin UI, find **Org Details**. Note your tenant name and the API base URL, which uses the format `https://{tenant}.api.identitynow.com`. Do not derive the base URL from the address that you use to open the user interface, because that address might be a vanity URL.
3. Choose your user name, and then choose **Preferences**.
4. Choose **Personal Access Tokens**, and then choose **New Token**.
5. Enter a description for the token.
6. Turn on the `sp:scopes:all` scope, and then choose **Create Token**.
7. Copy the Client ID and Client Secret for API authentication.

Authenticating with SailPoint ISC

The connector uses the OAuth 2.0 Client Credentials Grant flow. The pipeline exchanges the stored `client_id` and `client_secret` at the SailPoint token endpoint for a short-lived bearer access

token in JSON Web Token (JWT) format. SailPoint reports the remaining lifetime of each token in the `expires_in` field of the token response, and the connector refreshes the token before it expires. No user interaction is required after initial setup.

Open the AWS Secrets Manager console, choose **Store a new secret**, choose **Other type of secret**, and then add the following key-value pairs:

- `client_id` – Your SailPoint PAT Client ID.
- `client_secret` – Your SailPoint PAT Client Secret.

For example, name the secret `sailpoint-isc/pipeline-credentials`.

Configuring the CloudWatch pipeline

1. Create the destination CloudWatch Logs log group. In the CloudWatch console, choose **Logs**, **Log groups**, **Create log group**, and then enter a descriptive name. For example, use `/aws/cloudwatch/pipelines/sailpoint-isc`.
2. Create a source IAM role with the required permissions. For information about the required policies and permissions, see the section called “CloudWatch pipelines IAM policies and permissions”.
3. Open the CloudWatch console, navigate to **Pipelines**, and choose SailPoint ISC as the data source.
4. Configure OAuth 2.0 client credentials using the `client_id` and `client_secret` stored in AWS Secrets Manager.
5. Provide the SailPoint ISC API base URL and select the destination log group.
6. Verify the destination log group resource policy. For a new log group, the console creates the resource policy automatically. If the log group that you select already has a resource policy, the console warns you that the log group already has a resource policy and that you might need to modify the existing policy to add the permissions that pipelines need to write to the destination log group. In that case, update the existing resource policy within five minutes of pipeline creation to include the required pipeline write permissions. For more information, see the section called “Resource policies”.
7. Wait a few minutes for the initial data pull, check the destination log group for incoming events, and monitor pipeline metrics for successful ingestion.

> **Note**
>
> When you create the pipeline using the AWS CLI or API, a resource policy is never created automatically, regardless of whether the log group is new or existing. You must create the CloudWatch Logs resource policy manually before the pipeline becomes active.

Supported Open Cybersecurity Schema Framework event classes

This integration supports OCSF schema version v1.5.0. All SailPoint ISC audit events are retrieved from a single SailPoint ISC Search API endpoint and mapped to OCSF classes based on their event type category.

> **Note**
>
> Events that do not match an OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

Authentication (3002)

`AUTH` – Authentication events for Personal Access Token lifecycle operations, including token usage.

Account Change (3001)

- `PASSWORD_ACTIVITY` – Password change, reset, and policy enforcement events.
- `PROVISIONING` – Account provisioning and deprovisioning actions across connected sources.

Entity Management (3004)

- `USER_MANAGEMENT` – User lifecycle events including creation, modification, and deletion.
- `ACCESS_ITEM` – Access item creation, modification, and removal events.
- `CERTIFICATION` – Access certification campaign events including decisions and sign-offs.
- `SOURCE_MANAGEMENT` – Source connector configuration and management events.
- `SYSTEM_CONFIG` – System configuration and platform settings change events.
- `NON_EMPLOYEE` – Non-employee lifecycle management events.

- `IDENTITY_MANAGEMENT` – Identity profile and attribute management events.

User Access Management (3005)

`ACCESS_REQUEST` – Access request submission, approval, and denial events.

Ingestion-only event types

The following event types do not have an OCSF mapping and are forwarded to the sink without additional processing:

- `SSO` – Single sign-on events.
- `IAI_ADMIN_AUDIT` – AI-based identity analytics administrator audit events.

Known platform limitations

- **Search result paging limit** – The SailPoint Search API returns a maximum of 10,000 records for each page and limits offset-based paging to a total of 10,000 records for a single query. The connector uses one-hour time partitions so that each query window stays within the 10,000-record paging ceiling. High-volume tenants that generate more than 10,000 events in one hour might exceed the paging ceiling for that window.
- **API rate limits** – The limit is 100 requests per `client_id`, per API version, per 10 seconds, which is effectively 10 requests per second. When SailPoint returns an HTTP 429 response, the connector retries with progressively increasing delays. The limit applies per `client_id`, not per access token.
- **Token expiration** – SailPoint issues short-lived access tokens and reports the remaining lifetime of each token in the `expires_in` field of the token response. The connector refreshes the token before it expires, so no manual intervention is required.
- **Maximum PATs per user** – SailPoint supports a maximum of 10 Personal Access Tokens per user. Plan your token allocation if the same administrator user needs Personal Access Tokens for multiple integrations.
- **Event time-based filtering** – The connector collects events by the time in the `created` field, not by the time that SailPoint indexed the event. Because the connector queries a closed time window, an event that becomes searchable after the window that contains its `created` time was queried is not collected.

Troubleshooting

`401 Unauthorized`

The client credentials are invalid or expired. Verify the `client_id` and `client_secret` stored in AWS Secrets Manager, and generate a new PAT in SailPoint if necessary.

`403 Forbidden`

The PAT might have insufficient permissions. Make sure that the `sp:scopes:all` scope is turned on for the token, and that the token was generated by a user with the `ORG_ADMIN` user level. Create a new PAT with the correct user and scope if necessary.

`429 Too Many Requests`

The API rate limit of 10 requests per second for each `client_id` was exceeded. The connector automatically retries with progressively increasing delays. If the error persists, verify that other integrations are not sharing the same `client_id`.

No data appears in the log group

The pipeline resource policy might not have been added within five minutes. Delete and recreate the pipeline, and then immediately add the CloudWatch Logs resource policy.

`400 Bad Request`

The API base URL might be malformed or contain an incorrect tenant or domain. Verify that `base_url` uses the format `https://{tenant}.api.{domain}.com`. Check the tenant name in the SailPoint Admin UI under **Org Details**.

No events are returned despite valid credentials

- Verify that the PAT user has `ORG_ADMIN` privileges. Users with lower privilege levels might not have access to all audit event types.
- Check whether the tenant has events within the configured `range`. A newly provisioned tenant might not have historical data.

Some event types are missing

- Confirm that the `sp:scopes:all` scope is turned on for the PAT. Restricted scopes might limit which event types are accessible.

- Check whether the event type has an OCSF mapping. Event types that have no mapping, such as SS0, are still delivered to the sink, but they are delivered in their original form instead of as OCSF events.

Vanity URL confusion

Some SailPoint deployments use vanity URLs, such as `iga.acme.com`, for the user interface. The `base_url` must use the API endpoint format `https://{tenant}.api.{domain}.com`. Find your tenant name in the SailPoint Admin UI under **Org Details**. After you know your API host, you can call the OAuth information endpoint at `https://{tenant}.api.identitynow.com/oauth/info` to confirm the tenant ID and the OAuth endpoints for the tenant.

CloudWatch pipelines configuration for SailPoint ISC

The following pipeline configuration collects identity governance audit events from SailPoint ISC using OAuth 2.0 client credentials authentication. Replace the placeholder values shown between << and >> with values for your environment.

```
extension:
  aws:
    secrets:
      sailpoint_isc:
        secret_id: <<arn:aws:secretsmanager:<region>:<account-id>:secret:<secret-
name>>>
        region: <<region>>
        sts_role_arn: <<arn:aws:iam::<account-id>:role/<role-name>>>
pipeline:
  source:
    sailpoint_isc:
      base_url: "https://your-tenant.api.identitynow.com"
      authentication:
        oauth2:
          client_id: "${{aws_secrets:sailpoint_isc:client_id}}"
          client_secret: "${{aws_secrets:sailpoint_isc:client_secret}}"
      range: P90D
  sink:
    - cloudwatch_logs:
        log_group: <<sailpoint-isc-log-group>>
  processor:
    - ocsf:
        schema:
```

```
          sailpoint_isc: null
        version: "1.5"
        mapping_version: 1.5.0
```

Configure AWS Secrets Manager access for securely retrieving credentials at runtime. Define a named secret block, such as `sailpoint_isc`, with `secret_id` set to the ARN of your AWS Secrets Manager secret, `region` set to the AWS Region where the secret is stored, and `sts_role_arn` set to the IAM role ARN to assume when accessing the secret. Reference credentials in the pipeline using `${{aws_secrets:<name>:<key>}}`. For more information, see the AWS Secrets Manager extension.

Source parameters

`base_url` (required)

The SailPoint ISC API base URL. Use the format `https://{tenant}.api.{domain}.com`. The tenant name and domain are specific to your SailPoint deployment. Find the base URL in the SailPoint Admin UI under **Org Details**.

`range` (optional)

The lookback duration for audit events in ISO 8601 format. Minimum: `PT1H`. Maximum: `P90D`. Default: `P90D`.

`authentication.oauth2.client_id` (required)

The OAuth 2.0 Client ID from the SailPoint PAT. Store it in AWS Secrets Manager and reference it using `${{aws_secrets:sailpoint_isc:client_id}}`.

`authentication.oauth2.client_secret` (required)

The OAuth 2.0 Client Secret from the SailPoint PAT. Store it in AWS Secrets Manager and reference it using `${{aws_secrets:sailpoint_isc:client_secret}}`.

Note

Both `client_id` and `client_secret` must be stored in AWS Secrets Manager. Generate the credentials by creating a Personal Access Token in SailPoint ISC under **Preferences**, **Personal Access Tokens**. The PAT must be created by a user with `ORG_ADMIN` privileges, and the `sp:scopes:all` scope must be turned on, to provide access to all audit event types.

Sink parameters

`log_group` (required)

The name of the destination CloudWatch Logs log group, such as `sailpoint-isc-log-group`. For complete configuration options, see the CloudWatch Logs sink.

Processor parameters

`schema.sailpoint_isc` (required)

Set this parameter to `null`. The presence of the key activates the SailPoint ISC schema mapping and transforms ingested events into OCSF format.

`version` (required)

The OCSF schema version. Set this parameter to `"1.5"`.

`mapping_version` (required)

The mapping version. Set this parameter to `1.5.0`. For the full parameter reference, see the OCSF processor.

Salesforce ELF integration configuration

Salesforce is a cloud-based Customer Relationship Management (CRM) platform that provides business applications for sales, service, marketing, and IT operations. It generates detailed operational logs through Event Log Files (ELF) including login activity, API usage, report execution, and administrative changes, with hourly or daily granularity, along with audit trail events. CloudWatch pipelines use the Salesforce REST API to poll these logs for ingestion into CloudWatch Logs.

Topics

- Source configuration for Salesforce ELF
- CloudWatch pipelines configuration for Salesforce ELF

Source configuration for Salesforce ELF

Prerequisites

Before you begin, make sure you have the following:

- A Salesforce account with the EventLogFile feature enabled
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create and manage Amazon S3 buckets, AWS Secrets Manager secrets, and IAM roles

Integrating with Salesforce ELF

To integrate Salesforce ELF with CloudWatch Logs, you must configure a pipeline. First, set up a Salesforce account and an AWS Secrets Manager secret with required authentication credentials to receive data from Salesforce. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Salesforce ELF exports 50+ event types through EventLogFile (CSV) and SetupAuditTrail (JSON) using the Salesforce REST API, which delivers logs directly to the configured sink (CloudWatch Logs).

Instructions to set up Salesforce account

1. **Create Salesforce Account**
   - Go to the Salesforce developer sign-up page.
   - Register with your name, email, and a unique username (email format).
   - Verify your email and set your password.
   - Note your instance URL (for example, `https://yourorg-dev-ed.my.salesforce.com`) – this is the `instance_url` for pipeline configuration.
2. **Create an External Client App**
   - In **Setup**, search **External Client App Manager** in Quick Find.
   - Choose **New External Client App**.
   - Fill in **App Name** (for example, "CloudWatch-Collector") and **Contact Email**.
   - Under **API (Enable OAuth Settings)**, check **Enable OAuth**.
   - Set **Callback URL**: `https://login.salesforce.com/services/oauth2/callback`
   - Move these OAuth Scopes from Available to Selected: `Full access (full)`, `Manage user data through APIs (api)`, `Perform requests at any time (refresh_token, offline_access)`
   - Choose **Create**.
3. **Configure Policies**

- Open your app in **External Client App Manager**.
- **Policies** tab > **Edit**.
- **Permitted Users**: Select "Admin approved users are pre-authorized" > **OK**.
- **Select Profiles**: Add **System Administrator**.
- Under **OAuth Flows and External Client App Enhancements**: Check **Enable Client Credentials Flow**.
- **Run As (Username)**: Enter the System Administrator's username (find it under **Setup > Users**).
- Choose **Save**.

4. **Get Consumer Key and Secret**
   - Open your app > **Settings** tab > **OAuth Settings**.
   - Choose **Consumer Key and Secret** button (verify identity by email code if prompted).
   - Copy **Consumer Key** (`client_id`) and **Consumer Secret** (`client_secret`).
5. **Create and Assign Permission Set**
   - **Setup > Permission Sets** > **New** (for example, "CloudWatch-Collector-Permissions").
   - **System Permissions > Edit** – enable: **API Enabled**, **View Event Log Files**, **View Setup and Configuration**.
   - **Save**, then **Manage Assignments > Add Assignment** > select the Run-As user > **Assign**.
6. **Enable Event Log File**
   - **Setup > Event Monitoring**.
   - Enable the **Generate event log files** toggle.
7. **Store Credentials in AWS Secrets Manager**
   - In AWS Secrets Manager, choose **Store a new secret** > **Other type of secret**.
   - Add key/value pairs: `client_id` = Consumer Key, `client_secret` = Consumer Secret.
   - Name the secret (for example, `salesforce-credentials`) and copy the ARN.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Salesforce, choose `salesforce_elf` as the data source. Provide the `instance_url`, `client_id`, and `client_secret`. After creating the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following events. Events that are not listed but pulled are not mapped to OCSF and will be forwarded to the sink as raw logs.

Account Change (3001)

Contains the following events:

createduser, createdcustomersuccessuser, createdpartneruser, activateduser, unfrozeuser, changedpassword, resetpassword, deactivateduser, frozeuser, CSPUserDisabled, PRMUserDisabled, PermSetAssign, PermSetAssign_HasExpiration, PermSetGroupAssign, PermSetLicenseAssign, addeduserpackagelicense, granteduserpackagelicense, enabledForIntegrations, loginasgrantedtopartnerbt, loginasgrantedtosfdc, PermSetUnassign, PermSetUnassign_HasExpiration, PermSetGroupUnassign, PermSetLicenseUnassign, removeduserpackagelicense, revokeduserpackagelicense, disabledForIntegrations, loginasrevokedtopartnerbt, loginasrevokedtosfdc, insertTwoFactorInfo2, insertTwoFactorWebAuthN, insertAuthenticatorPairing, lightningloginenroll, insertTwoFactorTempCode, deleteTwoFactorInfo2, deleteTwoFactorWebAuthN, deleteAuthenticatorPairing, lightninglogincancel, unlockeduser

Authentication (3002)

Contains the following events:

- Login
- LoginAs
- Logout

API Activity (6003)

Contains the following events:

- ApiTotalUsage
- BulkApi
- BulkApi2
- CompositeApi

- CompositeApiSubrequest
- RestApi
- API
- ContinuationCalloutSummary
- ExternalCrossOrgCallout
- ExternalCustomApexCallout
- ExternalDataSourceCallout
- ExternalODataCallout
- NamedCredential
- MetadataApiOperation

Datastore Activity (6005)

Contains the following events:

- AsynchronousReportRun
- Dashboard
- DatabaseSave
- MultiblockReport
- Report
- ReportExport
- Search
- UniqueQuery

Detection Finding (2004)

Contains the following events:

- BlockedRedirect
- CorsViolation
- CspViolation
- HostnameRedirects

- InsecureExternalAssets
- InsufficientAccess
- TransactionSecurity

Entity Management (3004)

Contains SetupAuditTrail events for entity creation, modification, deletion, and configuration changes. This includes a large number of Salesforce administrative actions such as creating/deleting Apex classes, custom fields, permission sets, profiles, workflows, and system configuration toggles.

File Hosting Activity (6006)

Contains the following events:

- ContentDistribution
- ContentDocumentLink
- ContentTransfer
- DocumentAttachmentDownloads

HTTP Activity (4002)

Contains the following event:

- URI – user interaction with the web browser UI

CloudWatch pipelines configuration for Salesforce ELF

Collects EventLogFile and Setup AuditTrail events from Salesforce using OAuth2 authentication.

Configure the Salesforce ELF source with the following parameters:

```
source:
  salesforce_elf:
    instance_url: "<<your-instance-url>>"
    authentication:
      client_id: "${{aws_secrets:salesforce-credentials:client_id}}"
```

```
      client_secret: "${{aws_secrets:salesforce-credentials:client_secret}}"
```

Parameters

`instance_url` (required)

Salesforce instance URL. Must use HTTPS.

`authentication.client_id` (required)

OAuth2 Consumer Key. Supports `${{aws_secrets:...}}`.

`authentication.client_secret` (required)

OAuth2 Consumer Secret. Supports `${{aws_secrets:...}}`.

`range` (optional)

Set backfill duration in ISO 8601 format (for example, `PT24H`, `P7D`, `P30D`). When set, overwrites both `elf_backfill` and `setup_audit_trail_backfill` with the same duration. When omitted, `elf_backfill` defaults to `P30D` (30 days) and `setup_audit_trail_backfill` defaults to `P180D` (180 days).

Salesforce RTEM integration configuration

Salesforce is a cloud-based Customer Relationship Management (CRM) platform that provides business applications for sales, service, marketing, and IT operations. It generates real-time security events through the Pub/Sub API (gRPC) covering login activity, API usage, file events, and data access across 19+ event channels with sub-second delivery. CloudWatch pipelines use the Salesforce Pub/Sub API to stream these events into CloudWatch Logs.

Topics

- Source configuration for Salesforce RTEM
- CloudWatch pipelines configuration for Salesforce RTEM

Source configuration for Salesforce RTEM

Prerequisites

Before you begin, make sure you have the following:

- A Salesforce account with the Event Monitoring feature enabled
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create and manage Amazon S3 buckets, AWS Secrets Manager secrets, and IAM roles

Integrating with Salesforce RTEM

To integrate Salesforce RTEM with CloudWatch Logs, you must configure a pipeline. First, set up a Salesforce account and an AWS Secrets Manager secret with required authentication credentials to receive data from Salesforce. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Salesforce RTEM streams 19 default RTEM channels plus optional CDC and custom Platform Events through the Pub/Sub API, which delivers logs directly to the configured sink (CloudWatch Logs).

Instructions to set up Salesforce account

1. **Create Salesforce Account**
   - Go to the Salesforce developer sign-up page.
   - Register with your name, email, and a unique username (email format).
   - Verify your email and set your password.
   - Note your instance URL (for example, `https://yourorg-dev-ed.my.salesforce.com`) – this is the `instance_url` for pipeline configuration.
2. **Create an External Client App**
   - In **Setup**, search **External Client App Manager** in Quick Find.
   - Choose **New External Client App**.
   - Fill in **App Name** (for example, "CloudWatch-Collector") and **Contact Email**.
   - Under **API (Enable OAuth Settings)**, check **Enable OAuth**.
   - Set **Callback URL**: `https://login.salesforce.com/services/oauth2/callback`
   - Move these OAuth Scopes from Available to Selected: `Full access (full)`, `Perform requests at any time (refresh_token, offline_access)`, `Access Pub/Sub API (pubsub_api)`
   - Choose **Create**.
3. **Configure Policies**

- Open your app in **External Client App Manager**.
- **Policies** tab > **Edit**.
- **Permitted Users**: Select "Admin approved users are pre-authorized" > **OK**.
- **Select Profiles**: Add **System Administrator**.
- Under **OAuth Flows and External Client App Enhancements**: Check **Enable Client Credentials Flow**.
- **Run As (Username)**: Enter the System Administrator's username (find it under **Setup > Users**).
- Choose **Save**.

4. **Get Consumer Key and Secret**
   - Open your app > **Settings** tab > **OAuth Settings**.
   - Choose **Consumer Key and Secret** button (verify identity by email code if prompted).
   - Copy **Consumer Key** (`client_id`) and **Consumer Secret** (`client_secret`).
5. **Create and Assign Permission Set**
   - **Setup > Permission Sets** > **New** (for example, "CloudWatch-Collector-Permissions").
   - **System Permissions > Edit** – enable: **API Enabled**, **View Real-Time Event Monitoring Data**.
   - **Save**, then **Manage Assignments > Add Assignment** > select the Run-As user > **Assign**.
6. **Enable Event Storage and Streaming**

   If storage is not enabled for an event type, the Pub/Sub API subscription connects but receives zero events silently.
   - **Setup > Event Manager**.
   - For each event type, choose the dropdown > **Enable Storage and Streaming**.
7. **Find Your Organization ID**
   - **Setup > Company Information**.
   - Copy the **Salesforce.com Organization ID** (15 or 18 character alphanumeric, for example, `00D5e000003TIrB`).
8. **Store Credentials in AWS Secrets Manager**
   - In AWS Secrets Manager, choose **Store a new secret** > **Other type of secret**.
   - Add key/value pairs: `client_id` = Consumer Key, `client_secret` = Consumer Secret.
   - Name the secret (for example, `salesforce-credentials`) and copy the ARN.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Salesforce, choose `salesforce_rtem` as the data source. Provide the `instance_url`, `org_id`, and use the `client_id` and `client_secret`. After creating the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following events. Events not listed are not mapped to OCSF and are forwarded to the sink as raw logs.

Authentication (3002)

Covers the following events:

- LoginEventStream
- LoginAsEventStream
- LogoutEventStream

File Hosting Activity (6006)

Covers the following event:

- FileEvent

API Activity (6003)

Covers the following events:

- BulkApiEventStream
- ApiEventStream

Datastore Activity (6005)

Covers the following events:

- ReportEvent
- ListViewEventStream

CloudWatch pipelines configuration for Salesforce RTEM

Collects real-time events from Salesforce using OAuth2 authentication.

Configure the Salesforce RTEM source with the following parameters:

```
source:
  salesforce_rtem:
    instance_url: "<<your-instance-url>>"
    org_id: "<<your-org-id>>"
    authentication:
      client_id: "${{aws_secrets:salesforce-credentials:client_id}}"
      client_secret: "${{aws_secrets:salesforce-credentials:client_secret}}"
```

Parameters

`instance_url` (required)

Salesforce instance URL (for example, `https://myorg.my.salesforce.com`). Must use HTTPS.

`org_id` (required)

Salesforce organization ID (15 or 18 character alphanumeric).

`authentication.client_id` (required)

OAuth2 Consumer Key. Supports `${{aws_secrets:...}}`.

`authentication.client_secret` (required)

OAuth2 Consumer Secret. Supports `${{aws_secrets:...}}`.

`additional_topics` (optional)

Extra topics beyond the 19 defaults (CDC, custom Platform Events).

`replay_preset` (optional)

Controls where to start consuming events. Valid values: `LATEST`, `EARLIEST`. Default: `EARLIEST`.

SentinelOne Singularity Endpoint integration configuration

To integrate SentinelOne Singularity Endpoint with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your SentinelOne source by configuring Amazon S3 and

Amazon SQS to receive endpoint logs. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Topics

- Source configuration for SentinelOne
- CloudWatch pipelines configuration for SentinelOne

Source configuration for SentinelOne

Integrating with SentinelOne Singularity Endpoint

SentinelOne Singularity Endpoint is an AI-powered endpoint security platform that provides real-time protection against malware, ransomware, and zero-day attacks. It uses behavioral analysis and machine learning to detect and stop threats autonomously. The platform supports automated response, rollback, and threat remediation. It gives centralized visibility and control across all endpoints. CloudWatch pipelines enables you to collect this data in CloudWatch Logs.

Instructions to setup Amazon S3 and Amazon SQS

Configuring SentinelOne Singularity Endpoint to send logs to an Amazon S3 bucket involves several steps, primarily focused on setting up the Amazon S3 bucket, Amazon SQS queue, IAM roles, and then configuring the Amazon Telemetry Pipeline.

- Create Amazon S3 bucket that stores SentinelOne Singularity Endpoint logs.
- Configure Singularity Cloud Funnel or intermediate Syslog server with Amazon S3 bucket details to push logs.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring the CloudWatch Pipeline

To configure the pipeline to read logs, choose SentinelOne Singularity Endpoint as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and the SentinelOne Singularity Endpoint events that maps to File System Activity (1001), Process Activity (1007), HTTP Activity (4002) and DNS Activity (4003).

File System Activity contains the following events:

- MALICIOUSFILE
- FILECREATION
- FILEDELETION
- FILEMODIFICATION
- FILERENAME
- FILESCAN

Process Activity contains the following events:

- PROCESSCREATION
- PROCESSTERMINATION
- DUPLICATETHREAD
- REMOTETHREAD
- PROCESSMODIFICATION
- DUPLICATEPROCESS
- OPENPROCESS
- PROCESSINJECTION
- PROCESSMODIFIER
- PROCESSEXIT
- OPENPRIVILEGEDPROCESSFROMKERNEL

HTTP Activity contains the following events:

- HTTP

DNS Activity contains the following events:

- DNS

CloudWatch pipelines configuration for SentinelOne

The SentinelOne setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the SentinelOne source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "sentinelone_endpointsecurity"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "sentinelone_endpointsecurity".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

ServiceNow CMDB Audit Log

ServiceNow is an enterprise platform that provides IT service management (ITSM) and configuration management database (CMDB) capabilities for tracking and managing IT assets, configurations, and changes across organizations.

Topics

- Source configuration for ServiceNow CMDB Audit Log
- CloudWatch pipelines configuration for ServiceNow CMDB Audit Log

Source configuration for ServiceNow CMDB Audit Log

Integrating with ServiceNow CMDB

ServiceNow is an enterprise platform that provides IT service management (ITSM) and configuration management database (CMDB) capabilities for tracking and managing IT assets, configurations, and changes across organizations. CloudWatch Pipeline uses the ServiceNow Table API to retrieve information about sys_audit, syslog, sysevent, and syslog_transactions from your ServiceNow instance.

Authenticating with ServiceNow CMDB

To read the logs, the pipeline needs to authenticate with your ServiceNow instance. The ServiceNow Table API supports OAuth 2.0.

- Make sure the REST API is enabled on your ServiceNow instance.
- Enable OAuth 2.0 Client Credentials grant type in your ServiceNow instance
- Create an OAuth Application Registry for external client authentication
- In the AWS Secrets Manager, create a secret and store the Application (client) ID under the key `client_id` and the client secret under the key `client_secret`.
- Configure OAuth Application User and assign required roles

Configuring the CloudWatch Pipeline

When configuring the pipeline to read audit logs from ServiceNow choose ServiceNow CMDB as the data source. Fill in the required information like `instance_url` and the secret where `client_id` and `client_secret` are stored. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to Entity Management (3004), API Activity (6003), and Datastore Activity (6005). These events are from specific tables and filtered for CMDB reference.

Entity Management contains events from following tables:

- sys_audit

API Activity contains events from following tables:

- sysevent
- syslog

Datastore Activity contains events from following tables:

- syslog_transactions

CloudWatch pipelines configuration for ServiceNow CMDB Audit Log

Collects configuration management database (CMDB) data from ServiceNow using OAuth2 authentication.

Configure the ServiceNow CMDB source with the following parameters:

```
source:
  servicenow_cmdb:
    instance_url: "<example-instance-url>"
    range: "P7D"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`instance_url` (required)

ServiceNow instance URL.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for ServiceNow API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for ServiceNow API authentication.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

Slack Audit Log integration configuration

Slack is a cloud-based collaboration and messaging platform that enables team communication through channels, direct messages, file sharing, and integrations with external applications. CloudWatch pipelines use the Slack APIs (such as Events API and Audit Logs API) to retrieve information about user activities, messages, channel events, app interactions, and administrative actions across your Slack workspace.

Topics

- Source configuration for Slack Audit Log
- CloudWatch pipelines configuration for Slack Audit Log

Source configuration for Slack Audit Log

Integrating with Slack Audit Log

Slack is a cloud-based collaboration and messaging platform that enables team communication through channels, direct messages, file sharing, and integrations with external applications. CloudWatch pipelines use the Slack APIs (such as Events API and Audit Logs API) to retrieve information about user activities, messages, channel events, app interactions, and administrative actions across your Slack workspace. These APIs provide REST endpoints that allow access to event data, enabling the collection of communication and audit logs from your Slack environment for monitoring and analysis.

Authenticating with Slack

To read Slack audit logs, the pipeline needs to authenticate with your Slack workspace. The plugin supports OAuth API token authentication. Follow these instructions to get started with Slack APIs:

1. Before creating an app, register for the Slack Developer Program by visiting: `https://api.slack.com/developer-program`.
2. Log in to the Slack API portal and navigate to "Your Apps" then "Create New App." Choose "From scratch" and provide an app name and workspace. After creation, note down the Client ID and Client Secret from the "Basic Information" section.
3. Configure OAuth scopes under "OAuth & Permissions." Add required scopes such as `auditlogs:read`, `channels:read`, `groups:read`, `users:read`, and `channels:history` depending on your use case. Install the app to your workspace by choosing the install button in the Slack API portal. After installation, copy the User OAuth Token (starts with `xoxp-`). The `auditlogs:read` scope requires a user OAuth token.
4. In AWS Secrets Manager, create a secret that will hold the Slack token obtained in step 3. The secret's value must be a JSON object with a key that stores the token. Pick any secret name and any key name you prefer, but remember both — the pipeline configuration references them together using the syntax `${{aws_secrets:<secret-name>:<key-name>}}`.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read logs, choose Slack as the data source. Specify the range duration format (for example, PT21H for the last 21 hours) to control the time window of logs retrieved. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to Web Resources Activity (6001), Authentication (3002), Entity Management (3004), Account Change (3001), User Access Management (3005), File Hosting Activity (6006), and Detection Finding (2004). These events are sourced from the Slack AuditLogs API. Events that are not listed are not mapped to OCSF and will be forwarded to the sink as raw logs.

Web Resources Activity (6001) contains the following event types:

- private_channel_created
- private_channel_archive
- private_channel_converted_to_public
- private_channel_deleted
- private_channel_unarchive
- public_channel_archive
- public_channel_converted_to_private
- public_channel_created
- public_channel_deleted
- public_channel_preview
- public_channel_unarchive
- file_public_link_created
- file_public_link_revoked
- huddle_screenshare_on
- huddle_ended
- huddle_knock_accepted
- huddle_participant_dropped

- huddle_participant_joined
- huddle_participant_left
- huddle_started
- huddle_screenshare_off
- huddle_transcription_cancelled
- huddle_transcription_paused
- huddle_transcription_resumed
- huddle_transcription_started
- huddle_transcription_start_notification
- slack_ai_huddle_notes_generated
- list_linksharing_enabled
- canvas_linksharing_enabled
- list_linksharing_disabled
- canvas_linksharing_disabled

Entity Management (3004) contains the following event types:

- external_shared_channel_invite_accepted
- external_shared_channel_invite_approved
- external_shared_channel_invite_auto_revoked
- external_shared_channel_invite_created
- external_shared_channel_access_upgraded
- external_shared_channel_disconnect_and_archived
- external_shared_channel_disconnected
- external_shared_channel_invite_declined
- external_shared_channel_invite_expired
- external_shared_channel_invite_revoked
- channels_export_completed
- channels_export_deleted
- channels_export_downloaded

- channels_export_started
- scheduled_export_completed
- scheduled_export_deleted
- scheduled_export_downloaded
- scheduled_export_started
- manual_export_completed
- manual_export_deleted
- manual_export_downloaded
- manual_export_started
- manual_user_export_downloaded
- manual_user_export_completed
- manual_user_export_deleted
- manual_user_export_started
- external_shared_channel_connected

Account Change (3001) contains the following event types:

- role_change_to_owner
- role_change_to_admin
- role_change_to_guest
- role_change_to_user
- role_removed
- role_assigned
- guest_created
- guest_deactivated
- guest_reactivated
- user_created
- user_deactivated
- user_reactivated
- user_email_updated

- user_profile_updated
- user_profile_deleted
- guest_expiration_cleared
- guest_expiration_set
- guest_expired
- user_force_upgrade_non_compliant_mobile_app_version
- owner_transferred
- service_owner_transferred
- user_password_reset_requested
- user_password_reset_slack_security
- custom_tos_accepted
- user_session_settings_changed
- role_added_to_usergroup
- role_removed_from_usergroup

Authentication (3002) contains the following event types:

- bulk_session_reset_by_admin
- user_session_reset_by_admin
- user_logout_non_compliant_mobile_app_version
- user_logout
- user_logout_compromised
- user_login
- user_login_failed
- cli_login
- user_sessions_reset_by_anomaly_event_response
- user_session_invalidated

User Access Management (3005) contains the following event types:

- permissions_assigned

- user_channel_join
- guest_channel_join
- user_added_to_usergroup
- workflow_trigger_permission_added
- workflow_trigger_permission_set
- app_resources_granted
- app_resources_added
- app_scopes_expanded
- permissions_removed
- user_channel_leave
- guest_channel_leave
- user_removed_from_usergroup
- workflow_trigger_permission_removed
- role_modified_on_usergroup

File Hosting Activity (6006) contains the following event types:

- file_deleted
- file_download_blocked
- file_downloaded
- file_shared
- file_uploaded

Detection Finding (2004) contains the following event types:

- file_malicious_content_detected
- Anomaly

CloudWatch pipelines configuration for Slack Audit Log

Collects event logs from Slack using API token authentication.

Configure the Slack source with the following parameters:

```
source:
  slack_auditlogs:
    authentication:
        api_token: "${{aws_secrets:slack-account-credentials:apiToken}}"
    # Provide the time range (e.g., P7D for the last 7 days)
    range: "P7D"
```

Parameters

`authentication` (required)

Block containing Slack authentication settings. Contains the nested parameters listed in the following entry.

`api_token` (required)

Slack API token (`xoxp-...`) with the `auditlogs:read` scope. Typically sourced from AWS Secrets Manager using the `${{aws_secrets:<secret-name>:<key>}}` reference syntax.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

> **Note**
>
> The `api_token` value is retrieved from AWS Secrets Manager. The preceding parameter information can be obtained from the API credentials generated while setting up your Slack application.

Tanium integration configuration

Tanium is an endpoint management and security platform that provides real-time visibility and control across enterprise endpoints. CloudWatch pipelines collect detailed telemetry from Tanium including endpoint inventory, compliance posture, vulnerability findings, threat response alerts, and audit logs covering authentication, account changes, and group management activities.

Topics

- Source configuration for Tanium
- CloudWatch pipelines configuration for Tanium

Source configuration for Tanium

Integrating with Tanium

Tanium is an endpoint management and security platform that provides real-time visibility and control across enterprise endpoints. It collects detailed telemetry such as endpoint activity, security events, authentication events, and network activity. This data can be exported and integrated with AWS services for centralized monitoring and analytics. CloudWatch pipelines enable the ingestion of this data into CloudWatch Logs for further analysis and alerting.

Instructions to setup Tanium Connect, Amazon S3 and Amazon SQS

Prerequisites

- Access to the AWS Management Console with IAM administrative privileges.
- Access to the Tanium Console with permissions to configure Tanium Connect.

To set up Tanium Connect, Amazon S3, and Amazon SQS

1. In the AWS Console, create an Amazon S3 bucket in your desired region. Create separate prefixes (sub-folders) within the bucket for each log type to make sure clear log segregation and easier downstream processing.
2. Create an Amazon SQS Queue for Event Notifications. Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. Configure Amazon S3 Event Notifications on the bucket to send `s3:ObjectCreated:*` events to the Amazon SQS queue. This enables downstream consumers to be notified when new log files are delivered.
3. Create an IAM User with Programmatic Access. In IAM, create a new user with programmatic (CLI) access only. Attach a policy granting the minimum required permissions like `s3:PutObject`. Scope the policy to the target Amazon S3 bucket and prefixes. Generate an Access Key ID and Secret Access Key for the user. Store these credentials securely (for example, in a secrets manager). These will be needed in the following step.
4. Configure the Tanium Connect Destination. Log in to the Tanium Console. Navigate to Tanium Connect and create a new connection, select the type of log data and other general information. Select Amazon S3 as the destination. Enter the following details:

a. AWS Access Key ID

b. AWS Secret Access Key

c. S3 Bucket Name

d. AWS Region

e. Object key prefix (matching the sub-folder for the relevant log type)

5. Validate the Configuration. Save the connection and execute a test run. Verify that log data is successfully delivered to the expected Amazon S3 bucket and prefix. Confirm that the Amazon SQS queue is receiving event notifications for newly created objects.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Tanium, choose Tanium as the data source. Using Tanium Connect, data from Tanium custom and default reports is exported to Amazon S3, with event notifications delivered through Amazon SQS. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following report data and maps them to one of the 8 OCSF Classes: Device Inventory Info (5001), Vulnerability Finding (2002), Compliance Finding (2003), Detection Finding (2004), Authentication (3002), Account Change (3001), Entity Management (3004), and Group Management (3006). Out of all the custom reports pushed to S3 by the end user, the reports from the following listed datasources are transformed to the respective OCSF Classes. The rest of the data is ingested in raw format.

There is no specific event type in Tanium (except for audit logs). The following custom reports are generated by choosing the relevant modules in Tanium Connect:

- Tanium Platform
- Compliance Data – Comply Module
- Vulnerability Data – Comply Module
- Discovery Data – Discover Module
- Threat Response Alerts – Threat Response Module

The following sections describe the audit log mappings to OCSF event classes.

Authentication (3002)

From audit logs where `object_type_name` = "authentication":

- New Session Created
- New System User Session Created
- Authentication Failed
- User Logged Out
- User Failed to Log Out

Account Change (3001)

From audit logs where `object_type_name` = "api_token" or "user":

- `type_name` = CreateObject and `object_type_name` = api_token
- `type_name` = DeleteObject and `object_type_name` = api_token
- `type_name` = CreateObject and `object_type_name` = user
- `type_name` = DeleteObject and `object_type_name` = user
- `type_name` = UpdateObject and `object_type_name` = user

Entity Management (3004)

From audit logs where `reportName` = "TaniumThreatAuditLog":

- action = create
- action = update
- action = deleteBulk

Group Management (3006)

From audit logs where `object_type_name` = "user_group":

- action = UpdateObject and details containing "Assigned role" / "updated with role ids"
- action = UpdateObject and details containing "removed role"
- action = UpdateObject and details containing "user ids added"
- action = UpdateObject and details containing "user ids removed"

- action = UpdateObject and details containing "persona updated with user groups"
- action = DeleteObject
- action = CreateObject

CloudWatch pipelines configuration for Tanium

The Tanium setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the S3 source with the following parameters:

```
source:
  s3:
    acknowledgments: true
    sqs:
      queue_url: "https://sqs.us-east-1.amazonaws.com/1234567890/tanium-sqs"
      maximum_messages: '10'
      visibility_duplication_protection: true
      visibility_timeout: 180s
    aws:
      region: "us-east-1"
    notification_type: sqs
    notification_source: s3
    bucket_owners:
      tanium-test-logs-test: '1234567890'
    default_bucket_owner: '1234567890'
    codec:
      ndjson: {}
    compression: none
```

Parameters

`acknowledgments` (required)

Prevents data loss by only considering logs successfully processed after they are received by the sink. Set to `true` to enable.

`sqs.queue_url` (required)

The Amazon SQS queue URL that receives Amazon S3 event notifications when new objects land.

`sqs.maximum_messages` (optional)

Max number of Amazon SQS messages to pull per poll (batch size, up to Amazon SQS max of 10).

`sqs.visibility_duplication_protection` (optional)

Uses Amazon SQS visibility timeout to prevent duplicate processing of the same message by multiple workers.

`sqs.visibility_timeout` (optional)

How long (in seconds) a message stays invisible to other consumers after being picked up, giving the worker time to process it.

`aws.region` (required)

AWS region where the Amazon SQS queue and Amazon S3 bucket live.

`notification_type` (required)

`sqs` – Tells the plugin to expect notifications through Amazon SQS.

`notification_source` (optional)

`s3` – Indicates the Amazon SQS notifications originate from Amazon S3 event notifications.

`bucket_owners` (required)

Maps specific bucket names to their owning AWS account IDs, needed for cross-account access.

`default_bucket_owner` (required)

Fallback AWS account ID used for any bucket not explicitly listed in `bucket_owners`.

`codec` (required)

`ndjson` – Parses each Amazon S3 object as newline-delimited JSON (one JSON record per line).

`compression` (required)

none – Indicates the Amazon S3 objects are not compressed, so no decompression is applied before parsing.

Tenable Vulnerability Management integration configuration

Tenable Vulnerability Management is a cloud-based vulnerability management platform that provides continuous visibility into assets, vulnerabilities, compliance posture, and security findings.

CloudWatch pipelines use the Tenable REST API to retrieve asset inventories, vulnerability findings, compliance data, scan results, audit logs, and plugin metadata from your Tenable environment.

Topics

- Source configuration for Tenable Vulnerability Management
- CloudWatch pipelines configuration for Tenable Vulnerability Management

Source configuration for Tenable Vulnerability Management

Integrating with Tenable Vulnerability Management

CloudWatch pipelines use the Tenable REST API to retrieve asset inventories, vulnerability findings, compliance data, scan results, audit logs, and plugin metadata from your Tenable environment. The API exposes REST and bulk export endpoints that allow fetching security data for monitoring and analysis.

To integrate CloudWatch Pipelines with Tenable Vulnerability Management, complete the following high-level steps:

- Generate an Access Key and Secret Key from the Tenable UI.
- Store the credentials in AWS Secrets Manager.
- Create a CloudWatch pipeline with Tenable as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- An active Tenable account with the Administrator role
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager
- An AWS account with permissions to create and manage CloudWatch Logs log groups

Authenticating with Tenable Vulnerability Management

Tenable uses API key authentication with an access key and secret key. The pipeline passes these credentials through the `X-ApiKeys` header to authenticate each API request.

Configure Authentication for Tenable Vulnerability Management

To configure authentication credentials for the pipeline:

1. Log in to the Tenable Vulnerability Management UI.
2. Navigate to Settings > My Account > API Keys.
3. Generate a new API key pair. Note the Access Key and Secret Key values.
4. Make sure your account has the Administrator role.
5. Store the `access_key` and `secret_key` in AWS Secrets Manager.

Configuring the CloudWatch Pipeline

To configure the pipeline, choose Tenable as the data source. Provide the `access_key` and `secret_key`. After you create and activate the pipeline, vulnerability management data from Tenable will begin flowing into the selected CloudWatch Logs log group.

The following optional parameters are available:

- **Export settings** – chunk`_size` (default 1000, range 100–10,000), `export_backfill` (default `P30D`, range 30 minutes–90 days), `export_polling_interval` (default `P1D`, range 30 minutes–7 days)
- **Backfill durations** – Configure historical backfill for audit logs, plugins, and scans independently.
- **Polling intervals** – Configure independent polling intervals for audit, scan, and plugin streams.
- **Page sizes** – Configure the number of records retrieved per API request page.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports the following OCSF event classes:

- **Device Inventory Info [5001]** – Assets
- **Compliance Finding [2003]** – Compliance
- **Vulnerability Finding [2002]** – Vulnerability
- **Scan Activity [6007]** – Scans
- **Authentication [3002]** – Audit Logs (authentication-related actions)
- **API Activity [6003]** – Audit Logs (API-related actions)

- **Account Change [3001]** – Audit Logs (account modification actions)

Plugins are passed as raw data and are not mapped to OCSF event classes.

CloudWatch pipelines configuration for Tenable Vulnerability Management

Collects vulnerability management data from Tenable using API key authentication.

Configure the Tenable Vulnerability Management source with the following parameters:

```
source:
  tenable_vulnerabilitymanagement:
    authentication:
      access_key: "${{aws_secrets:<secret-name>:access_key}}"
      secret_key: "${{aws_secrets:<secret-name>:secret_key}}"
    range: "P30D"
```

Parameters

`authentication.access_key` (required)

The Tenable API access key, stored in AWS Secrets Manager.

`authentication.secret_key` (required)

The Tenable API secret key, stored in AWS Secrets Manager.

`range` (optional)

The historical time period for backfilling data. Uses ISO 8601 duration format. Minimum is `P1D`, maximum is `P90D`. Default is `P30D`.

Wazuh Platform integration configuration

Wazuh is an open-source security platform that provides unified XDR and SIEM capabilities including threat detection, integrity monitoring, incident response, and compliance across on-premises, cloud, and hybrid environments. The Wazuh Platform is the central component built on OpenSearch that stores and indexes security alerts, providing near real-time search and analytics through a RESTful API. Use CloudWatch pipelines with the Wazuh Platform API to retrieve security alerts, vulnerability findings, system inventory, and agent monitoring data.

Topics

- Source configuration for Wazuh Platform
- CloudWatch pipelines configuration for Wazuh Platform

Source configuration for Wazuh Platform

Integrating with Wazuh Platform

Wazuh Platform is an open-source security monitoring platform that delivers threat detection, file integrity monitoring, vulnerability assessment, and compliance across endpoints and cloud workloads. Use CloudWatch pipelines with the Wazuh Platform API (an Elasticsearch-compatible REST API) to retrieve security alerts, vulnerability findings, system inventory, and agent monitoring data from your self-hosted Wazuh Platform instance. The Indexer API provides search endpoints that you can use to access event data stored in daily-rotated indices, so you can collect security logs from your Wazuh Platform environment for centralized monitoring and analysis.

To integrate CloudWatch Pipelines with Wazuh Platform, complete the following high-level steps:

- Obtain the Wazuh Indexer host URL and credentials.
- Store the username and password in AWS Secrets Manager.
- Create a CloudWatch pipeline with Wazuh Platform as the data source.
- Verify that data is flowing into the pipeline.

Prerequisites

Before you begin, make sure you have the following:

- The Wazuh Indexer installed and running. Follow the official quickstart guide to install: Quickstart - Wazuh documentation
- Wazuh Indexer accessible over HTTPS (port 9200 or 443 with reverse proxy)
- Network connectivity from the pipeline to the Wazuh Indexer host (if behind a firewall, allow the pipeline's egress IPs)
- Valid credentials with Basic Auth (username/password) that have read permissions on the target indices
- User role permissions with at minimum `indices:data/read/search` on the target index patterns (for example, `wazuh-alerts-*`, `wazuh-states-*`)

- A sortable `timestamp` field present in the documents for incremental ingestion and `search_after` pagination (for inventory indices this may need to be added manually)
- If using self-signed TLS certificates, the pipeline must trust the CA
- An AWS account with permissions to create and manage CloudWatch Pipelines
- An AWS account with permissions to create, retrieve, and update secrets in AWS Secrets Manager

Authenticating with Wazuh Platform

To read Wazuh Platform data, the pipeline needs to authenticate with your Wazuh Indexer instance. The plugin supports HTTP Basic Authentication (username/password).

Follow these steps to configure authentication:

1. Ensure your Wazuh Indexer is installed and running. The Indexer runs on port 9200 over HTTPS. Verify access by running: `curl -k -u <username>:<password> https://<wazuh-host>:9200/`
2. Identify or create a user with read access to the required indices (`wazuh-alerts-*`, `wazuh-monitoring-*`, `wazuh-states-inventory-*`). The default `admin` user has full access. For production, create a dedicated read-only user through the Security plugin API.
3. In AWS Secrets Manager, create a secret that will hold the Wazuh Indexer credentials. The secret's value must be a JSON object with keys for `username` and `password`. For example, create a secret named `wazuh-credentials` with the following JSON value:

   ```
   {"username": "admin", "password": "your-wazuh-indexer-password"}
   ```

   The corresponding pipeline references are `${{aws_secrets:wazuh-credentials:username}}` and `${{aws_secrets:wazuh-credentials:password}}`.
4. Ensure network connectivity from the pipeline to the Wazuh Indexer host on port 9200/443.
5. If using self-signed TLS certificates (default for Wazuh), configure the pipeline to trust the Wazuh CA certificate, or disable certificate verification in the pipeline configuration.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read logs, choose Wazuh Platform as the data source. Provide the Host URL (your Wazuh Indexer endpoint including port 9200/443) and Authentication credentials.

The pipeline will poll for new data at the configured interval. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and transforms the following events. Events that are not listed but pulled are not mapped to OCSF and will be forwarded to the sink as raw logs.

Authentication (3002)

Contains authentication success and failure events including SSH, PAM, and Windows logon events.

Vulnerability Finding (2002)

Contains Vulnerability Detection findings including CVE findings from vulnerability scanning.

File System Activity (1001)

Contains File Integrity Monitoring events including file addition, modification, and deletion.

Device Inventory Info (5001)

Contains System Inventory and Agent monitoring events.

Detection Finding (2004)

Contains Rootcheck events generated for malware detection.

Application Lifecycle (6002)

Contains Container Security events.

Compliance Finding (2003)

Contains SCA compliance events.

Remediation Activity (7001)

Contains Active Response events.

API Activity (6003)

Contains Cloud Security events from Amazon, Azure, GCP, GitHub, and Office365.

CloudWatch pipelines configuration for Wazuh Platform

Collects audit logs from Wazuh Platform using Basic authentication.

Configure the Wazuh Platform source with the following parameters:

```
source:
  wazuh_platform:
    host: "https://<wazuh-indexer-host>:9200"
    authentication:
      username: "${{aws_secrets:wazuh-credentials:username}}"
      password: "${{aws_secrets:wazuh-credentials:password}}"
    acknowledgments: true
```

Parameters

`host` (required)

Wazuh Indexer base URL including scheme and port (for example, `https://<wazuh-indexer-host>:9200`). This is a customer-configured domain – each customer hosts their own Wazuh Indexer instance.

`authentication` (required)

Block containing Wazuh Indexer authentication settings. Contains the nested parameters listed below.

`authentication.username` (required)

Wazuh Indexer username for HTTP Basic Authentication. Typically sourced from AWS Secrets Manager using the `${{aws_secrets:<secret-name>:<key>}}` reference syntax.

`authentication.password` (required)

Wazuh Indexer password for HTTP Basic Authentication. Typically sourced from AWS Secrets Manager using the `${{aws_secrets:<secret-name>:<key>}}` reference syntax.

`acknowledgments` (optional)

Prevents data loss by only considering logs successfully processed after they are received by the sink. Accepts `true` or `false`. Default: `true`.

> **Note**
>
> The `username` and `password` values are retrieved from AWS Secrets Manager. The above parameter information can be obtained from the Wazuh Indexer installation or user management configuration.

WIZ CNAPP integration configuration

To integrate WIZ CNAPP with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your WIZ source by authenticating with OAuth2 and configuring API access. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Topics

- Source configuration for WIZ CNAPP
- CloudWatch pipelines configuration for WIZ

Source configuration for WIZ CNAPP

Integrating with Wiz CNAPP

Wiz is a cloud-native application protection platform (CNAPP) that provides comprehensive visibility and security across multi-cloud environments. CloudWatch Pipeline uses the Wiz GraphQL API to retrieve information about security posture, vulnerabilities, misconfigurations, threats, and audit activities from your cloud infrastructure. The Wiz GraphQL API enables access to security data through flexible GraphQL queries, allowing retrieval of audit logs, issues, vulnerability findings, configuration findings, and detections from the Wiz platform.

Authenticating with Wiz CNAPP

To read Wiz Cnapp audit Logs, the pipeline needs to authenticate with your account. The plugin supports OAuth2 Authentication. Follow these instructions to get started.

- Create a Service Account in Wiz with appropriate permissions. You must be logged in as a Wiz user with Write (W) permission on service accounts.
- Configure the Service Account and get newly created Client ID and Client Secret.

- In the AWS Secrets Manager, create a secret and store the Application (client) ID under the key `client_id` and the client secret under the key `client_secret`.
- Configure API permissions (scopes) for your Service Account.

  Required scope: `read:issues`, `read:detections`, `read:cloud_events_cloud`, `read:cloud_events_sensor`, `read:security_scans`, `read:vulnerabilities`, `read:cloud_configuration`, `admin:audit`
- Identify your GraphQL API endpoint: To find your specific endpoint check Tenant Info in the Wiz portal. The Wiz GraphQL API endpoint is `https://api.<region>.app.wiz.io/graphql`, where `<region>` corresponds to your Wiz tenant's datacenter (for example, us1, us2, eu1, eu2).

Configuring the CloudWatch Pipeline

When configuring the pipeline to read audit logs from Wiz, choose Wiz CNAPP as the data source. Fill in the required information like Region. After you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to Detection Finding (2004), Vulnerability Finding (2002), Compliance Finding (2003), Authentication (3002), and API Activity (6003).

Detection Finding contains all events from following sources:

- Issues
- Detections

Vulnerability Finding contains all events from following sources:

- Vulnerability Findings

Compliance Finding contains all events from following sources:

- Cloud Configuration Findings

Authentication contains events from following sources and given actions:

- Audit logs
- DeviceLogin
- Login

API Activity contains events from following sources and given actions:

- Audit logs
- AddSecurityScan
- AddSupportTicketContext
- AiAssistantSendMessage
- ApproveCopyResourceForensicsSettings...
- AssociateServiceTicket
- CancelReportRun
- ClearUIUserPreferences
- CompleteAuthMigration
- ConvertGitHubAppRegistrationCode
- CopyResourceForensicsToExternalAccount
- CreateActionTemplate
- CreateApplicationServiceDiscoveryRule
- CreateAutomationRule
- CreateCICDScanPolicy
- CreateCloudConfigurationFindingNote
- CreateCloudConfigurationRule
- CreateCloudConfigurationRules
- CreateCloudEventRule
- CreateComputeGroupTagsSet
- CreateConnector
- CreateControl
- CreateCustomIPRange
- CreateDashboard

- CreateDashboardWidget
- CreateDataClassifier
- CreateDigitalTrustCustomDomain
- CreateFileIntegrityMonitoringExclusion
- CreateHostConfigurationAssessmentNote
- CreateHostConfigurationRule
- CreateIgnoreRule
- CreateImageIntegrityValidator
- CreateIntegration
- CreateIssueNote
- CreateMalwareExclusion
- CreateMonitoredMetric
- CreateOutpost
- CreateOutpostCluster
- CreatePolicyPackage
- CreatePortalView
- CreateProject
- CreateRemediationAndResponseDeployment
- CreateRemediationPullRequest
- CreateReport
- CreateRuntimeResponsePolicy
- CreateSAMLIdentityProvider
- CreateSAMLUser
- CreateSavedCloudEventFilter
- CreateSavedGraphQuery
- CreateScannerAPIRateLimit
- CreateSecurityFramework
- CreateServiceAccount
- CreateSupportTicket

- CreateTestNode
- CreateUser
- CreateUserRole
- CreateVulnerabilityFindingNote
- DeleteActionTemplate
- DeleteApplicationServiceDiscoveryRule
- DeleteAutomationRule
- DeleteCICDScan
- DeleteCICDScanPolicy
- DeleteCloudConfigurationFindingNote
- DeleteCloudConfigurationRule
- DeleteCloudEventRule
- DeleteComputeGroupTagsSet
- DeleteConnector
- DeleteControl
- DeleteCustomIPRange
- DeleteDashboard
- DeleteDashboardWidget
- DeleteDataClassifier
- DeleteDigitalTrustCustomDomain
- DeleteFileIntegrityMonitoringExclusion
- DeleteHostConfigurationAssessmentNote
- DeleteHostConfigurationRule
- DeleteIgnoreRule
- DeleteImageIntegrityValidator
- DeleteIntegration
- DeleteIssueNote
- DeleteMalwareExclusion
- DeleteMonitoredMetric

- DeleteOutpost
- DeleteOutpostCluster
- DeletePolicyPackage
- DeletePortalView
- DeleteProject
- DeleteRemediationAndResponseDeployment
- DeleteReport
- DeleteRuntimeResponsePolicy
- DeleteSAMLIdentityProvider
- DeleteSavedCloudEventFilter
- DeleteSavedGraphQuery
- DeleteScannerAPIRateLimit
- DeleteSecurityFramework
- DeleteSecurityScan
- DeleteServiceAccount
- DeleteTestNode
- DeleteUser
- DeleteUserRole
- DeleteVulnerabilityFindingNote
- DisassociateServiceTicket
- DuplicateDashboard
- DuplicateDataClassifier
- DuplicateHostConfigurationRule
- DuplicateSecurityFramework
- DuplicateUserRole
- FinalizeCICDScan
- FinalizeCICDScanTelemetry
- GenerateWizContainerRegistryToken
- GraphSearch

- InitiateCICDScanTelemetry
- InitiateDiskScanContainerImage
- InitiateDiskScanDirectory
- InitiateDiskScanVirtualMachine
- InitiateDiskScanVirtualMachineImage
- InitiateIACScan
- InvokeOutpostClusterUpdate
- LegalConsent
- MergeDiscoveredApplicationService
- MigrateUsers
- ModifySAMLIdentityProviderGroupMappings
- ModifySAMLIdentityProviderPortalView...
- PromoteDiscoveredApplicationService
- ProvideAiFeedback
- ProvideAiGraphQueryExample
- ProvideAiGraphQueryFeedback
- ProvideIssueFeedback
- ReassessIssue
- RefreshResponseActions
- RegisterAgent
- ReportIDEActivityHeartbeat
- ReportIDEAnalytics
- RequestConnectorEntityScan
- RequestConnectorScan
- RerunReport
- ResetUserPassword
- RevokeSessions
- RevokeUserSessions
- RotateServiceAccountSecret
- RunAllControls

- RunCloudConfigurationRule
- RunControl
- RunControlsIntegrationAction
- RunIssuesIntegrationAction
- RunOutpostClusterUpdate
- RunResponseAction
- SAMLUserInitialProvision
- SendUserEmailInvite
- TagCICDScan
- TokenDeviceRefresh
- TokenRefresh
- UninstallOutpost
- UpdateAiSettings
- UpdateApplicationServiceDiscoveryRule
- UpdateAutomationRule
- UpdateBasicAuthSettings
- UpdateCICDScanPolicy
- UpdateChampionCenterJourneyItem
- UpdateCloudConfigurationFinding
- UpdateCloudConfigurationRule
- UpdateCloudConfigurationRules
- UpdateCloudCostSettings
- UpdateCloudEventRule
- UpdateCloudEventRules
- UpdateCloudEventSettings
- UpdateComputeGroupTagsSet
- UpdateConnector
- UpdateContainerRegistryCustomScannin...
- UpdateContainerRegistryGlobalScannin...
- UpdateControl

- UpdateControls
- UpdateCopyResourceForensicsSettings
- UpdateCustomIPRange
- UpdateCustomIPRangesSettings
- UpdateCustomUserRolesSettings
- UpdateDashboard
- UpdateDashboardSettings
- UpdateDashboardWidget
- UpdateDataClassifier
- UpdateDataFinding
- UpdateDataScannerSettings
- UpdateDigitalTrustCustomDomain
- UpdateDigitalTrustDashboardSettings
- UpdateDigitalTrustSAMLIdentityProvider
- UpdateDiscoveredApplicationServices
- UpdateEventTriggeredScanningSettings
- UpdateExternalExposureScannerSettings
- UpdateExternalExposureSettings
- UpdateFileIntegrityMonitoringExclusion
- UpdateFileIntegrityMonitoringSettings
- UpdateForensicsPackageSettings
- UpdateGraphEntity
- UpdateHostConfigurationRule
- UpdateHostConfigurationRuleAssessment
- UpdateHostConfigurationRules
- UpdateIPRestrictions
- UpdateIgnoreRule
- UpdateImageIntegrityValidator
- UpdateIntegration
- UpdateInternalExposureSettings

- UpdateIssue
- UpdateIssueNote
- UpdateIssueSettings
- UpdateIssues
- UpdateKubernetesGlobalScanningConfig...
- UpdateLoginSettings
- UpdateMalwareExclusion
- UpdateMonitoredMetric
- UpdateMonitoredMetricSettings
- UpdateNode
- UpdateNonOSDiskScanningSettings
- UpdateNotificationSettings
- UpdateOutpost
- UpdateOutpostCluster
- UpdatePolicyPackage
- UpdatePortalInactivityTimeoutSettings
- UpdatePortalSettings
- UpdatePortalView
- UpdatePreviewHubItem
- UpdateProject
- UpdateRemediationAndResponseDeployment
- UpdateReport
- UpdateReportSettings
- UpdateRepositorySettings
- UpdateResponseAction
- UpdateResponseActions
- UpdateRuntimeResponsePolicy
- UpdateSAMLIdentityProvider
- UpdateSavedCloudEventFilter
- UpdateSavedGraphQuery

- UpdateScannerAPIRateLimit
- UpdateScannerExclusionSettingsConstr...
- UpdateScannerExclusionSettingsTimeLi...
- UpdateScannerExclusionSizeLimits
- UpdateScannerExclusionTags
- UpdateScannerResourceTagSettings
- UpdateScannerResourceTags
- UpdateScannerSettings
- UpdateSecretInstance
- UpdateSecurityFramework
- UpdateSecurityScan
- UpdateServiceAccount
- UpdateSessionLifetimeSettings
- UpdateSupportContactList
- UpdateSystemHealthIssue
- UpdateSystemHealthIssues
- UpdateTechnology
- UpdateTenantNewsletterSettings
- UpdateUIUserPreferences
- UpdateUser
- UpdateUserRole
- UpdateUserSelectedPortalView
- UpdateVersionControlOrganizationSett...
- UpdateVersionControlRepositorySettings
- UpdateViewerPreferences
- UpdateVulnerability
- UpdateVulnerabilityAssessmentSettings
- UpdateVulnerabilityFinding
- UpdateVulnerabilityFindingStatus
- UpsertAgentTelemetry

CloudWatch pipelines configuration for WIZ

Collects cloud-native application protection platform (CNAPP) data from Wiz using OAuth2 authentication.

Configure the Wiz CNAPP source with the following parameters:

```
source:
  wiz_cnapp:
    region: "<example-region>"
    range: "P7D"
    authentication:
      oauth2:
        client_id: "${{aws_secrets:<secret-name>:client_id}}"
        client_secret: "${{aws_secrets:<secret-name>:client_secret}}"
```

Parameters

`region` (required)

Wiz region for your organization.

`authentication.oauth2.client_id` (required)

OAuth2 client ID for Wiz API authentication.

`authentication.oauth2.client_secret` (required)

OAuth2 client secret for Wiz API authentication.

`range` (optional)

The time range for log collection. Uses ISO 8601 duration format (for example, `P7D` for the last 7 days, `PT21H` for 21 hours). Default is 0 hours, and the maximum is 90 days.

Zscaler Internet Access

CloudWatch pipelines enables you to collect logs from Zscaler Internet Access (ZIA). ZIA is a cloud-based secure web gateway that protects users connecting to the internet.

Topics

- Source configuration for Zscaler Internet Access

- Pipeline configuration for Zscaler Internet Access

Source configuration for Zscaler Internet Access

Integrating with Zscaler Internet Access

Zscaler Internet Access (ZIA) is a cloud-based secure web gateway that protects users connecting to the internet. It inspects all internet traffic to block malware, phishing, and data leaks using advanced threat detection and SSL inspection. ZIA enforces security policies in real time without requiring on-premises hardware. It ensures safe and compliant internet access for users anywhere. CloudWatch pipelines enables you to collect this data in CloudWatch Logs.

Instructions to setup Amazon S3 and Amazon SQS

Configuring ZIA to send logs to an Amazon S3 bucket involves several steps, primarily focused on setting up the Amazon S3 bucket, Amazon SQS queue, IAM roles, and then configuring the Amazon Telemetry Pipeline.

- Create Amazon S3 bucket that stores ZIA logs and create separate folders for each log type. Create IAM user and grant s3 write permission, console access not needed only CLI and create Access key and Secret key for this account.
- Configure NSS feeds with Amazon S3 bucket details to push logs.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Zscaler Internet Access, choose Zscaler Internet Access (ZIA) as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to DNS Activity (4003), HTTP Activity (4002), Network Activity (4001), and Authentication (3002). Each event comes from a source as mentioned below.

DNS Activity covers all events from source:

- DNS Logs

HTTP Activity covers all events from source:

- Web Logs

Network Activity covers all events from source:

- Firewall Logs

Authentication covers events from source:

- Admin Audit Logs - Event actions: SIGN_IN, SIGN_OUT

Pipeline configuration for Zscaler Internet Access

The Zscaler setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Zscalar source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "zscaler_internetaccess"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "zscaler_internetaccess".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

Zeek integration configuration

Zeek is an open-source network security monitoring platform widely used for analyzing network traffic and generating detailed logs about network activities across an organization's infrastructure. It passively monitors network traffic and provides deep visibility into communications by producing structured logs for multiple network protocols and security-relevant events. CloudWatch pipelines allow ingestion of Zeek log data into CloudWatch Logs, providing scalable collection, processing, normalization, and integration with downstream AWS security and monitoring services.

Topics

- Source configuration for Zeek
- CloudWatch pipelines configuration for Zeek

Source configuration for Zeek

Integrating with Zeek

To integrate Zeek with CloudWatch Logs, you must configure both the source and the pipeline. First, set up your Zeek source by configuring Amazon S3 and Amazon SQS to receive data. Then, configure the CloudWatch pipeline to ingest the data from your source into CloudWatch Logs.

Instructions to setup Amazon S3 and Amazon SQS

Configuring Zeek with Fluent Bit to send logs to an Amazon S3 bucket involves several steps, primarily focused on setting up the Amazon S3 bucket, Amazon SQS queue, IAM roles, and then configuring the CloudWatch Pipeline.

Configuring Zeek logs using Fluent Bit

- Install Fluent Bit (a lightweight log collector that reads log files and forwards them to destinations such as Amazon S3) on the Zeek host and configure it to tail Zeek log files (for example, `/opt/zeek/logs/current/*.log`).
- Configure AWS credentials (IAM role or `aws configure`) so Fluent Bit has permission to upload objects to the Amazon S3 bucket.
- Update the Fluent Bit configuration to use the S3 output plugin, specifying the bucket name, Region, and S3 key path for Zeek logs.
- Start and enable the Fluent Bit service to continuously collect Zeek logs and upload them to Amazon S3 for downstream ingestion.

Amazon S3 and Amazon SQS configuration

- Amazon S3 bucket that stores the Zeek logs should reside in the same AWS Region.
- Configure the Amazon S3 bucket to create event notifications, specifically for "Object Create" events. These notifications should be sent to an Amazon SQS queue.
- Create an Amazon SQS queue in the same AWS Region as your Amazon S3 bucket. This queue will receive notifications when new log files are added to the Amazon S3 bucket.

Configuring the CloudWatch Pipeline

When configuring the pipeline to read data from Zeek, choose Zeek as the data source. After filling in the required information and you create the pipeline, data will be available in the selected CloudWatch Logs log group.

Supported Open Cybersecurity Schema Framework Event Classes

This integration supports OCSF schema version v1.5.0 and events that map to multiple OCSF classes. The following table lists the supported event mappings.

Event name	OCSF class
conn	Network Activity (4001)
dns	DNS Activity (4003)
http	HTTP Activity (4002)
ssl	Network Activity (4001)
ssh	SSH Activity (4007)
kerberos	Authentication (3002)
rdp	RDP Activity (4005)
files	Network Activity (4001)
notice	Detection Finding (2004)
known_hosts	Base Event (0)
x509	Network Activity (4001)
ftp	FTP Activity (4008)
smtp	Email Activity (4009)
dhcp	DHCP Activity (4004)
ntlm	Authentication (3002)

Event name	OCSF class
smb_files	SMB Activity (4006)
smb	SMB Activity (4006)
dce_rpc	SMB Activity (4006)
ldap	Authentication (3002)
ldap_search	Network Activity (4001)
quic	Network Activity (4001)
tunnel	Tunnel Activity (4014)
pe	Base Event (0)
weird	Base Event (0)
known_services	Base Event (0)
software	Software Inventory Info (5020)
reporter	Base Event (0)

Events that do not match any OCSF mapping transformation are automatically passed through and sent directly to the configured sink without additional processing.

CloudWatch pipelines configuration for Zeek

The Zeek setup on AWS reads log data from Amazon S3 buckets using Amazon SQS notifications for new object events.

Configure the Zeek source using the following parameters:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
```

```
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "zeek"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name` (required)

Identifies the data source. This can be any string value that represents your data source. Example: "zeek".

`aws.region` (required)

The AWS Region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports csv, json, ndjson codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

on_error (optional)

Determines how to handle errors in Amazon SQS. Can be either retain_messages or delete_messages. Default is retain_messages.

Custom log data from CloudWatch Logs or an Amazon S3 bucket

You can create pipelines for custom data sources using the following approaches:

1. **CloudWatch Custom Logs** – Define pipelines on your existing CloudWatch custom log groups by providing:
   - A data source name
   - A data source type

   For more information on data source name and type, see the CloudWatch Logs User Guide.
2. **CloudWatch Logs Syslog service** – Process Syslog messages delivered to a CloudWatch Logs log group. Before creating the pipeline, complete Setting up syslog ingestion and verify that messages are arriving in the destination log group.

   In the console, choose **CloudWatch Logs syslog service** and select RFC 5424, RFC 3164, Cisco, or F5. For API or AWS CLI configuration, use the cloudwatch_logs source with syslog as the data source name and the corresponding data source type: rfc5424, rfc3164, cisco, or f5_bigip.

   ```
   source:
     cloudwatch_logs:
       aws:
         sts_role_arn: "arn:aws:iam::123456789012:role/MyCloudWatchLogsRole"
       log_event_metadata:
         data_source_name: "syslog"
         data_source_type: "rfc5424"
   ```

   A pipeline matches one data source name and type. Create a separate pipeline for each Syslog format that you want to process. Syslog messages are parsed during ingestion, so pipeline processors are optional.

   For the shared source parameter reference and restrictions, see the section called “AWS service logs from CloudWatch Logs”. For pipeline IAM requirements, see the section called “Custom and Syslog data from CloudWatch Logs”.

3. **S3 Custom Sources** – Process arbitrary logs stored in S3 buckets by configuring source for the pipeline:

```
source:
  s3:
    aws:
      region: "us-east-1"
      sts_role_arn: "arn:aws:iam::<account>:role/<role-name>"
    compression: "gzip"
    codec:
      ndjson:
    data_source_name: "my_custom_logs"
    default_bucket_owner: "123456789012"
    bucket_owners:
      my-bucket: "123456789012"
    disable_bucket_ownership_validation: false
    notification_type: "sqs"
    sqs:
      queue_url: "https://sqs.region.amazonaws.com/<account>/<queue-name>"
    on_error: "retain_messages"
```

> **Note**
>
> The CSV processor can't be used with S3 custom sources. Instead, use the CSV codec in the S3 source.

Parameters

`notification_type` (required)

Specifies the notification mechanism. Must be "sqs" to use SQS for S3 event notifications.

`data_source_name`

Identifies the data source. This can be any string value that represents your data source. Example: "my_custom_logs".

`aws.region` (required)

The AWS region where the S3 bucket and SQS queue are located.

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing S3 and SQS resources.

`codec` (required)

Codec configuration for parsing S3 objects. Supports `csv`, `json`, `ndjson` codecs.

`compression` (optional)

Compression type of the S3 objects. Valid values are "none", "gzip", "automatic". Defaults to "none".

`sqs.queue_url` (required for SQS)

The complete SQS queue URL that receives S3 bucket notifications when new objects are created.

`on_error` (optional)

Determines how to handle errors in Amazon SQS. Can be either `retain_messages` or `delete_messages`. Default is `retain_messages`.

Custom source configuration

When creating a pipeline for custom sources:

- A parser must be the first processor in the pipeline if the data source is CloudWatch Logs
- You can specify any supported processor for custom log pipelines

Topics

- Configuring Custom S3 Bucket Sources

Configuring Custom S3 Bucket Sources

With CloudWatch pipelines, you can process arbitrary logs stored in S3 buckets.

Prerequisites

To use Amazon S3 as the source for a pipeline, first create an S3 bucket. For instructions, see Creating a general purpose bucketin the *Amazon S3 User Guide*.

> **Note**
>
> If the S3 bucket used as a source in the pipeline is in a different AWS account, you also need to enable cross-account read permissions on the bucket. This allows the pipeline to read and process the data. To enable cross-account permissions, see Bucket owner granting cross-account bucket permissions in the *Amazon S3 User Guide*.
> If your S3 buckets are in multiple accounts, use a `bucket_owners` map. For an example, see Cross-account S3 access in the *OpenSearch* documentation.

To set up S3-SQS processing, you also need to perform the following steps:

- Create an Amazon SQS queue.
- Enable event notifications on the S3 bucket with the SQS queue as a destination.

Configure the pipeline role

Unlike other source plugins that push data to a pipeline, the S3 source plugin has a read-based architecture in which the pipeline pulls data from the source. Therefore, in order for a pipeline to read from S3, you must specify a role within the pipeline's S3 source configuration that has access to both the S3 bucket and the Amazon SQS queue. The pipeline will assume this role in order to read data from the queue.

You can find example role permissions in Source-specific IAM policies. Note that this role must have a trust relationship with the CloudWatch pipelines service principle. You can find an example trust policy configuration for your pipeline role in Trust relationships.

Create the pipeline

After you've set up your permissions, you can configure a pipeline depending on your Amazon S3 use case.

Select **Create pipeline** in the **Pipelines** tab under **Ingestion** in the CloudWatch Console. Follow the pipeline wizard steps and provide the SQS queue ARN and required pipeline IAM role when prompted. Optionally provide a data source name and type to attach to the log group destination in CloudWatch Logs.

Be sure to configure a CloudWatch Logs resource policy if one isn't already configured to the destination log group and then select **Create pipeline** in the **Review and create** step. The pipeline will be created and data will begin to flow within 5 minutes if successful.

Amazon S3 cross account as a source

You can grant access across accounts with Amazon S3 so that CloudWatch pipelines can access S3 buckets in another account as a source. To enable cross-account access, see Bucket owner granting cross-account bucket permissions in the *Amazon S3 User Guide*. After you have granted access, make sure that your pipeline role has the required permissions.

Then, you can create a pipeline using `bucket_owners` to enable cross-account access to an Amazon S3 bucket as a source.

Custom source configuration

When creating a pipeline for custom sources:

- A parser must be the first processor in the pipeline
- You can specify any supported processor for custom log pipelines

AWS service logs from CloudWatch Logs

Intercepts log events from CloudWatch Logs that match the log event metadata configuration.

> **Important**
>
> Pipelines with processors mutate the log events in the original CloudWatch log group they are intercepted from for logs from AWS services. To preserve unmodified copies of your log data, enable the `include_original` option in the `cloudwatch_logs` sink configuration, or use the **Keep original log** toggle in the console. For more information, see the section called “Sinks”.

Configuration

Configure the CloudWatch Logs source with the following parameters:

```
source:
```

```
  cloudwatch_logs:
    aws:
      sts_role_arn: "arn:aws:iam::123456789012:role/MyCloudWatchLogsRole"
    log_event_metadata:
      data_source_name: "<data_source_name>"
      data_source_type: "<data_source_type>"
```

Parameters

`aws.sts_role_arn` (required)

The ARN of the IAM role to assume for CloudWatch Logs interception.

`log_event_metadata.data_source_name` (required)

Identifies the specific AWS service that generated the log events or a custom log source name. For custom logs, this can be any string up to 15 characters when `data_source_type` is "default".

`log_event_metadata.data_source_type` (required)

Specifies the category or type of logs within the AWS service, or "default" for custom logs. Set to "default" to enable custom log source names.

For more information on data source name and type, see the CloudWatch Logs User Guide.

Restrictions

The following restrictions apply to CloudWatch Logs sources:

- No two pipelines can use the `cloudwatch_logs` source with identical `data_source_name` and `data_source_type` metadata criteria.
- When `data_source_name` is `aws_cloudtrail`, only empty processors (`[]`) or the `ocsf` processor are allowed.

CloudWatch pipelines processors

CloudWatch pipelines processors transform, parse, and enrich log data as it flows through the pipeline. Log pipelines support up to 20 processors that are applied sequentially in the order they are defined. Metrics pipelines support up to 20 processors, applied sequentially in the order you define them. For metrics pipeline processors, see the section called “Metrics processors”.

Transformation metadata

When a pipeline processes log events, CloudWatch pipelines automatically adds transformation metadata to each processed log entry. This metadata indicates that the log has been transformed, making it easy to distinguish between original and processed data. If you enable the **Keep original log** option during pipeline creation, you can compare the original log with the transformed version at any time.

The following table describes the processor categories.

Category	Description
Parsers	Convert raw log data into structured formats, such as Open Cybersecurity Schema Framework (OCSF), CSV, JSON, and so on
Transformers	Modify log data structure; add, copy, move, or delete fields
String Processors	Manipulate string values; case conversion, trimming, substitution
Metrics processors	Add, delete, rename attributes and metrics on OTel metric datapoints

Topics

- Parser processors
- Transformation processors
- String manipulation processors
- Filter processors
- Common processor use cases
- Processor compatibility and restrictions
- Metrics pipeline processors

Parser processors

Parser processors convert raw or semi-structured log data into structured formats. Each pipeline can have at most one primary parser processor, which must be the first processor in the pipeline.

The XML parser is an exception: it operates on fields produced by a primary parser and you can add at most 5 instances to a single pipeline.

Conditional processing not supported

Parser processors (except Grok and XML) do not support conditional processing with the `when` parameter. This includes OCSF, CSV, JSON, KeyValue, VPC, Route53, RDS, WAF, Postgres, and Amazon CloudFront parsers. For more information, see the section called "Expression syntax for conditional processing".

OCSF processor

Parses and transforms log data according to Open Cybersecurity Schema Framework (OCSF) standards.

Configuration

Configure the OCSF processor with the following parameters:

```
processor:
  - ocsf:
      version: "1.5"
      mapping_version: 1.5.0
      schema:
          microsoft_office365_management_activity:
```

Parameters

`version` (required)

The OCSF schema version to use for transformation. Must be 1.5

`mapping_version` (required)

The OCSF mapping version for transformation. Must be 1.5.0.

`schema` (required)

Schema object specifying the data source type. The supported schemas depend on the pipeline source type - each source type has its own set of compatible OCSF schemas. You must use a schema that matches your pipeline's source type.

This table lists the supported schema combinations.

Pipeline source type	Supported Schemas	Version	Mapping Version
cloudwatch_logs	cloud_trail:	1.5	Not required
cloudwatch_logs	route53_resolver:	1.5	Not required
cloudwatch_logs	vpc_flow:	1.5	Not required
cloudwatch_logs	eks_audit:	1.5	Not required
cloudwatch_logs	aws_waf:	1.5	Not required
cloudwatch_logs	aws_nlb:	1.5	Not required
s3	Any OCSF schema	Any	Any
microsoft_office365	microsoft_office365:	1.5	1.5.0
microsoft_entraid	microsoft_entraid:	1.5	1.5.0
microsoft_windows_event	microsoft_windows_event:	1.5	1.5.0
paloaltonetworks_nextgenerationfirewall	paloaltonetworks_nextgenerationfirewall:	1.5	1.5.0
okta_auth0	okta_auth0:	1.5	1.5.0
okta_sso	okta_sso:	1.5	1.5.0
crowdstrike_falcon	crowdstrike_falcon:	1.5	1.5.0

Pipeline source type	Supported Schemas	Version	Mapping Version
github_auditlogs	github_auditlogs:	1.5	1.5.0
sentinelone_endpointsecurity	sentinelone_endpointsecurity:	1.5	1.5.0
servicenow_cmdb	servicenow_cmdb:	1.5	1.5.0
wiz_cnapp	wiz_cnapp:	1.5	1.5.0
zscaler_internetaccess	zscaler_internetaccess:	1.5	1.5.0

CSV processor

Parses CSV formatted data into structured fields.

Configuration

Configure the CSV processor with the following parameters:

```
processor:
  - csv:
      column_names: ["col1", "col2", "col3"]
      delimiter: ","
      quote_character: '"'
```

Parameters

`column_names` (optional)

Array of column names for parsed fields. Maximum 100 columns, each name up to 128 characters. If not provided, defaults to column_1, column_2, and so on.

`delimiter` (optional)

Character used to separate CSV fields. Must be a single character. Defaults to comma (,).

`quote_character` (optional)

Character used to quote CSV fields containing delimiters. Must be a single character. Defaults to double quote (").

To use the processor without specifying additional parameters, use the following command:

```
processor:
  - csv: {}
```

Grok processor

Parses unstructured data using Grok patterns. At most 1 Grok is supported per pipeline. For details on the Grok transformer in CloudWatch Logs see Processors that you can use in the *CloudWatch Logs User Guide*.

Configuration

Configure the Grok processor with the following parameters:

When the data source is a dictionary, you can use this configuration:

```
processor:
  - grok:
      match:
       source_key: ["%{WORD:level} %{GREEDYDATA:msg}"]
```

When the data source is CloudWatch Logs, you can use this configuration:

```
processor:
  - grok:
      match:
       source_key: ["%{WORD:level} %{GREEDYDATA:msg}"]
```

Parameters

`match` (required)

Field mapping with Grok patterns. Only one field mapping allowed.

`match.<field>` (required)

Array with single Grok pattern. Maximum 512 characters per pattern.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

> **Important**
>
> If the Grok processor is used as the parser (first processor) in a pipeline and its `when` condition evaluates to false, the entire pipeline does not execute for that log event. Parsers must run for downstream processors to receive structured data.

VPC processor

Parses VPC Flow Log data into structured fields.

Configuration

Configure the VPC processor with the following parameters:

```
processor:
  - parse_vpc: {}
```

JSON processor

Parses JSON data into structured fields.

Configuration

Configure the JSON processor with the following parameters:

```
processor:
  - parse_json:
      source: "message"
      destination: "parsed_json"
```

Parameters

`source` (optional)

The field containing the JSON data to parse. If omitted, the entire log message is processed

`destination` (optional)

The field where the parsed JSON will be stored. If omitted, parsed fields are added to the root level

Route 53 processor

Parses Route 53 resolver log data into structured fields.

Configuration

Configure the Route 53 processor with the following parameters:

```
processor:
  - parse_route53: {}
```

Amazon RDS processor

Parses Amazon RDS Aurora log data into structured fields. The `parse_rds` processor is supported only when the pipeline's `data_source_name` is `amazon_rds`. It applies the parsing logic that matches the pipeline's `data_source_type`.

Configuration

Configure the Amazon RDS processor with the following parameters:

```
processor:
  - parse_rds: {}
```

Key-value processor

Parses key-value pair formatted data into structured fields.

Configuration

Configure the key-value processor with the following parameters:

```
processor:
  - key_value:
      source: "message"
      destination: "parsed_kv"
      field_delimiter: "&"
      key_value_delimiter: "="
```

Parameters

`source` (optional)

Field containing key-value data. Maximum 128 characters.

`destination` (optional)

Target field for parsed key-value pairs. Maximum 128 characters.

`field_delimiter` (optional)

Pattern to split key-value pairs. Maximum 10 characters.

`key_value_delimiter` (optional)

Pattern to split keys from values. Maximum 10 characters.

`overwrite_if_destination_exists` (optional)

Whether to overwrite existing destination field.

`prefix` (optional)

Prefix to add to extracted keys. Maximum 128 characters.

non_match_value (optional)

Value for keys without matches. Maximum 128 characters.

To use the processor without specifying additional parameters, use the following command:

```
processor:
  - key_value: {}
```

XML parser

Use the XML parser to convert a specified field that contains an XML string to JSON format. Use the XML parser when your log events contain embedded XML fields that you want to query as

structured data. The XML parser operates on a named field that already contains an XML string. Place this parser after a primary parser in the pipeline. You can add at most 5 `parse_xml` parsers to a single pipeline.

Configuration

Configure the XML parser with the following parameters:

```
processor:
  - parse_json:
      source: "@message"
  - parse_xml:
      source: "body"
      destination: "parsed_xml"
```

Parameters

`source` (Required)

Specifies the field that contains the XML string to parse. Use dot notation to access nested fields. For example, `event.body`. Maximum 128 characters.

`destination` (Optional)

Specifies the field where the parsed XML structure is stored. If you omit this parameter, the parsed fields are added to the root level. Maximum 128 characters.

`when` (Optional)

A conditional expression that determines whether this parser runs. Maximum length is 256 characters. For more information, see the section called “Expression syntax for conditional processing”.

Example – XML parser output

Given the following JSON log event with an embedded XML field:

```
{
  "body": "<Person id=\"123\" active=\"true\"><name>John</name><age>30</age></Person>"
}
```

With the following configuration:

```
processor:
  - parse_json:
      source: "@message"
  - parse_xml:
      source: "body"
      destination: "parsed_xml"
```

The XML parser produces the following output:

```
{
  "body": "<Person id=\"123\" active=\"true\"><name>John</name><age>30</age></Person>",
  "parsed_xml": {
    "id": "123",
    "active": "true",
    "name": "John",
    "age": "30"
  }
}
```

Behavior notes

Nesting depth

The XML parser supports a maximum nesting depth of 25 levels. Elements nested beyond this limit produce an error.

Error handling

Malformed XML does not fail the pipeline. The parser preserves the original `@message` and sets `@pipeline.processing.status = "error"` on the event.

Transformation processors

Transformation processors modify the structure of log events by adding, copying, moving, or removing fields.

add_entries processor

Adds static key-value pairs to log events. At most 1 `add_entries` processor can be added to a pipeline.

Configuration

Configure the add_entries processor with the following parameters:

```
processor:
  - add_entries:
      entries:
        - key: "environment"
          value: "production"
          overwrite_if_key_exists: false
```

Parameters

`entries` (required)

Array of key-value pairs to add to each log event.

`entries[].key` (required)

The field name to add to the log event. Supports nested fields using dot notation.

`entries[].value` (required)

The static value to assign to the key.

`entries[].overwrite_if_key_exists` (optional)

Boolean flag that determines behavior when the key already exists. Defaults to false.

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider.

Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

copy_values processor

Copies values from one field to another. At most 1 `copy_values` processor can be added to a pipeline.

Configuration

Configure the copy_values processor with the following parameters:

```
processor:
  - copy_values:
      entries:
        - from_key: "user_id"
          to_key: "backup_user"
          overwrite_if_to_key_exists: false
```

Parameters

`entries` (required)

Array of copy operations to perform on each log event.

`entries[].from_key` (required)

The field name to copy the value from. Uses dot notation for nested fields.

`entries[].to_key` (required)

The field name to copy the value to. Will create nested structures if using dot notation.

`entries[].overwrite_if_to_key_exists` (optional)

Boolean flag controlling behavior when target field already exists. Defaults to false.

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

delete_entries processor

Removes specified fields from log events.

Configuration

Configure the delete_entries processor with the following parameters:

```
processor:
  - delete_entries:
      with_keys: ["temp_field", "debug_info"]
```

Parameters

`with_keys` (required)

Array of field names to remove from each log event. Supports nested field deletion using dot notation.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

move_keys processor

Moves fields from one location to another.

Configuration

Configure the move_keys processor with the following parameters:

```
processor:
  - move_keys:
      entries:
        - from_key: "old_field"
          to_key: "new_field"
          overwrite_if_to_key_exists: true
```

Parameters

`entries` (required)

Array of move operations. Maximum 5 entries.

`entries[].from_key` (required)

Source field name. Maximum 128 characters.

`entries[].to_key` (required)

Target field name. Maximum 128 characters.

`entries[].overwrite_if_to_key_exists` (optional)

Whether to overwrite existing target field.

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

flatten processor

Flattens nested object structures.

Configuration

Configure the flatten processor with the following parameters:

```
processor:
  - flatten:
      source: "metadata"
      target: "flattened"
      remove_processed_fields: true
      exclude_keys: ["sensitive_data"]
```

Parameters

`source` (required)

Field containing nested object to flatten.

`target` (required)

Target field prefix for flattened keys.

`remove_processed_fields` (optional)

Whether to remove the original nested field after flattening.

`exclude_keys` (optional)

Array of keys to exclude from flattening. Maximum 20 keys, each up to 128 characters.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

lookup processor

Enriches log events with data from a CloudWatch Logs lookup table. The processor matches fields in your log events against fields in the lookup table and appends specified fields to your log events. Use this processor for data enrichment scenarios such as mapping user IDs to user details, product codes to product information, or error codes to error descriptions. At most 1 `lookup` processor can be added to a pipeline.

> **Note**
>
> If a lookup table is used in a pipeline, you must provide an execution role with `logs:GetLookupTable` permissions on the table. For more information, see the section called “CloudWatch pipelines IAM policies and permissions”.

Configuration

Configure the lookup processor with the following parameters:

```
processor:
  - lookup:
      lookup_table: "arn:aws:logs:us-east-1:123456789012:lookup-table:my_lookup_table"
      match_keys:
        - log_key: "src_ip"
          lookup_key: "ip_address"
      entries:
        - source: "hostname"
          target: "src_hostname"
          overwrite_if_exists: true
          default_value: "UNKNOWN"
```

Parameters

`lookup_table` (required)

The ARN of the CloudWatch Logs lookup table to use for enrichment. Maximum length is 2048 characters.

`match_keys` (required)

Array of key pairs that define how to match log event fields to lookup table fields. Minimum 1, maximum 5 match keys. When multiple match keys are specified, a lookup table row must match all keys to produce a result (AND logic).

`match_keys[].log_key` (required)

The field name in the log event to match against. Maximum 128 characters.

`match_keys[].lookup_key` (required)

The column name in the lookup table to match against. Maximum 128 characters.

`entries` (required)

Array of fields to add to the log event from the matching lookup table row. Minimum 1, maximum 10 entries.

`entries[].source` (required)

The column name in the lookup table to retrieve the value from. Maximum 128 characters.

`entries[].target` (optional)

The field name to add to the log event. If not specified, the `source` column name is used as the field name. Maximum 128 characters.

`entries[].overwrite_if_exists` (optional)

Boolean flag that determines behavior when the target field already exists in the log event. Defaults to false.

`entries[].default_value` (optional)

The value to use when the lookup table has no matching row. If not specified, no field is added to the log event on a miss. Minimum length is 1, maximum length is 512 characters.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

Example

Consider a lookup table named `network_assets` with the following rows:

ip_address	hostname	owner	location
10.0.1.12	web-server-01	team-alpha	us-east-1
10.0.2.45	db-server-03	team-beta	us-west-2
10.0.3.78	cache-node-07	team-alpha	eu-west-1

Given the following log event:

```
{
  "timestamp": "2026-05-04T12:00:00Z",
  "src_ip": "10.0.2.45",
  "action": "connection_opened",
  "bytes": 2048
}
```

And the following processor configuration:

```
processor:
  - lookup:
      lookup_table: "arn:aws:logs:us-east-1:123456789012:lookup-table:network_assets"
      match_keys:
        - log_key: "src_ip"
          lookup_key: "ip_address"
      entries:
        - source: "hostname"
          target: "src_hostname"
        - source: "owner"
          default_value: "unassigned"
        - source: "location"
          target: "src_region"
          overwrite_if_exists: false
```

The processor produces the following enriched log event:

```
{
  "timestamp": "2026-05-04T12:00:00Z",
  "src_ip": "10.0.2.45",
  "action": "connection_opened",
  "bytes": 2048,
  "src_hostname": "db-server-03",
  "owner": "team-beta",
  "src_region": "us-west-2"
}
```

IAM permissions

When a pipeline uses the lookup processor, the pipeline's execution role must include `logs:GetLookupTable` permission for the referenced table. The following example policy statement grants this permission:

```
{
  "Effect": "Allow",
  "Action": "logs:GetLookupTable",
  "Resource": "arn:aws:logs:<region>:<account-id>:lookup-table:<table-name>"
}
```

geoip processor

The geoip processor enriches log events with geographic location data based on IP addresses. The processor uses MaxMind databases to resolve IP addresses to geographic information such as city, country, continent, coordinates, and autonomous system number (ASN) details. Use this processor for network traffic analysis, user location enrichment, and security investigation scenarios.

Configuration

Configure the geoip processor with the following parameters:

```
processor:
  - geoip:
      entries:
        - source: "client_ip"
          target: "client_geo"
          include_fields:
            - "city_name"
            - "country_name"
            - "country_iso_code"
            - "continent_name"
            - "latitude"
            - "longitude"
          when: '.client_ip != "127.0.0.1"'
```

Parameters

`entries` (required)

Array of entry objects that define which IP fields to enrich with geographic data. Minimum 1, maximum 10 entries.

`entries[].source` (required)

The field name in the log event that contains the IP address to look up. Must reference a valid IPv4 or IPv6 address. Maximum 128 characters.

`entries[].target` (required)

The field name where the processor writes geographic enrichment data in the log event. Maximum 128 characters.

`entries[].include_fields` (required)

Array of geographic fields to include in the enrichment result. Minimum 1, maximum 12 fields. The following fields are available:

- `continent_code` – Two-letter continent code (for example, "NA").
- `continent_name` – Name of the continent (for example, "North America").
- `country_name` – Name of the country (for example, "United States").
- `country_iso_code` – ISO 3166-1 alpha-2 country code (for example, "US").
- `city_name` – Name of the city (for example, "Seattle").
- `postal_code` – Postal code associated with the IP address location.
- `time_zone` – IANA time zone name (for example, "America/Los_Angeles").
- `latitude` – Latitude coordinate of the IP address location.
- `longitude` – Longitude coordinate of the IP address location.
- `network` – CIDR notation of the network associated with the IP address.
- `asn` – Autonomous system number associated with the IP address.
- `asn_organization` – Organization name associated with the autonomous system number.

`entries[].when` (optional)

Conditional expression that determines whether this entry executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

`when` (optional)

Top-level conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

Example

The following example shows a sample log event with IP address fields:

```
{
  "timestamp": "2026-05-04T12:00:00Z",
```

```
  "src_ip": "203.0.113.50",
  "dst_ip": "198.51.100.25",
  "action": "ALLOW",
  "bytes": 4096
}
```

The following processor configuration maps both source and destination IP fields to geographic enrichment targets:

```
processor:
  - geoip:
      entries:
        - source: "src_ip"
          target: "src_geo"
          include_fields:
            - "city_name"
            - "country_name"
            - "country_iso_code"
            - "latitude"
            - "longitude"
            - "asn"
            - "asn_organization"
        - source: "dst_ip"
          target: "dst_geo"
          include_fields:
            - "country_name"
            - "country_iso_code"
```

The processor produces the following enriched log event:

```
{
  "timestamp": "2026-05-04T12:00:00Z",
  "src_ip": "203.0.113.50",
  "dst_ip": "198.51.100.25",
  "action": "ALLOW",
  "bytes": 4096,
  "src_geo": {
    "city_name": "Seattle",
    "country_name": "United States",
    "country_iso_code": "US",
    "latitude": 47.6062,
    "longitude": -122.3321,
    "asn": 16509,
```

```
    "asn_organization": "Amazon.com, Inc."
  },
  "dst_geo": {
    "country_name": "Germany",
    "country_iso_code": "DE"
  }
}
```

String manipulation processors

String processors modify text values within log events through operations like case conversion, trimming, and pattern matching.

lowercase_string processor

Converts specified fields to lowercase.

Configuration

Configure the lowercase_string processor with the following parameters:

```
processor:
  - lowercase_string:
      with_keys: ["status", "method"]
```

Parameters

`with_keys` (required)

Array of field names to convert to lowercase. Only processes string values.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

uppercase_string processor

Converts specified fields to uppercase.

Configuration

Configure the uppercase_string processor with the following parameters:

```
processor:
  - uppercase_string:
      with_keys: ["status_code", "method"]
```

Parameters

`with_keys` (required)

Array of field names to convert to uppercase. Only processes string values.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

trim_string processor

Removes leading and trailing whitespace from specified fields.

Configuration

Configure the trim_string processor with the following parameters:

```
processor:
  - trim_string:
      with_keys: ["message", "user_input"]
```

Parameters

`with_keys` (required)

Array of field names to trim whitespace from. Only processes string values.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

substitute_string processor

Performs string substitution using regular expressions.

Configuration

Configure the substitute_string processor with the following parameters:

```
processor:
  - substitute_string:
      entries:
        - source: "message"
          from: "ERROR"
          to: "WARN"
```

Parameters

`entries` (required)

Array of substitution operations to perform on each log event.

`entries[].source` (required)

The field to perform string substitution on.

`entries[].from` (required)

The regular expression pattern to match and replace.

`entries[].to` (required)

The replacement string for matched patterns.

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

truncate processor

Truncates field values to specified length.

Configuration

Configure the truncate processor with the following parameters:

```
processor:
  - truncate:
      source_keys: ["message", "description"]
      length: 100
      start_at: 0
```

Parameters

`source_keys` (required)

Array of field names to truncate. Each field name maximum 128 characters.

`length` (optional)

Maximum length after truncation. Range: 1-8192.

`start_at` (optional)

Starting position for truncation. Range: 0-8192. Defaults to 0.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

extract_value processor

Extracts values using regular expressions.

Configuration

Configure the extract_value processor with the following parameters:

```
processor:
  - extract_value:
      entries:
```

```
        - source: "message"
          target: "extracted_data"
          from: "user=(?<user>\\w+)"
          to: "${user}"
          target_type: "string"
```

Parameters

`entries` (required)

Array of extraction operations. Maximum 20 entries.

`entries[].source` (required)

Field to extract from. Maximum 128 characters.

`entries[].target` (required)

Target field for extracted value. Maximum 128 characters.

`entries[].from` (required)

Regular expression pattern. Maximum 128 characters.

`entries[].to` (required)

Replacement pattern with capture groups. Maximum 128 characters.

`entries[].target_type` (optional)

Target data type ("integer", "double", "string", "boolean").

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider.

Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

convert_entry_type processor

Converts field values between different data types.

Configuration

Configure the convert_entry_type processor with the following parameters:

```
processor:
  - convert_entry_type:
      key: "count"
      type: "integer"
```

Parameters

`key` (required)

Single field name to convert.

`type` (required)

Target data type. Options: "integer", "double", "string", "boolean".

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

date processor

Parses and formats date/time fields.

Configuration

Configure the date processor with the following parameters:

```
processor:
  - date:
      match:
        - key: "timestamp"
```

```
          patterns: ["yyyy-MM-dd'T'HH:mm:ss.SSSSSS'Z'"]
      destination: "@timestamp"
      source_timezone: "UTC"
      destination_timezone: "America/New_York"
```

Parameters

`match` (required)

Array of date matching configurations. Maximum 10 entries.

`match[].key` (required)

Field containing the date string. Maximum 128 characters.

`match[].patterns` (required)

Array of date format patterns to try. Maximum 5 patterns, each up to 256 characters.

`destination` (optional)

Single target field for all parsed dates. Maximum 128 characters.

`source_timezone` (optional)

Source timezone for parsing.

`destination_timezone` (optional)

Target timezone for output.

`output_format` (optional)

Output date format. Maximum 64 characters.

`destination_type` (optional)

Output type - "timestampz", "long", or "string".

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

dissect processor

Extracts structured data using pattern matching.

Configuration

Configure the dissect processor with the following parameters:

```
processor:
  - dissect:
      map:
        message: "%{timestamp} %{level}"
```

Parameters

`map` (required)

Field mapping with dissect patterns.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

list_to_map processor

Converts array fields to map structures.

Configuration

Configure the list_to_map processor with the following parameters:

```
processor:
  - list_to_map:
      source: "tags"
      key: "name"
      value_key: "value"
      target: "tag_map"
```

Parameters

`source` (required)

Field containing array data. Maximum 128 characters.

`key` (required)

Field name to use as map key. Maximum 128 characters.

`value_key` (optional)

Field name to use as map value. Maximum 128 characters.

`target` (optional)

Target field for map structure. Maximum 128 characters.

`flatten` (optional)

Whether to flatten the resulting map.

`flattened_element` (optional)

Which element to use when flattening ("first" or "last").

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

rename_keys processor

Renames fields in log events.

Configuration

Configure the rename_keys processor with the following parameters:

```
processor:
  - rename_keys:
      entries:
        - from_key: "old_name"
          to_key: "new_name"
          overwrite_if_to_key_exists: true
```

Parameters

`entries` (required)

Array of rename operations. Maximum 5 entries.

`entries[].from_key` (required)

Current field name. Maximum 128 characters.

`entries[].to_key` (required)

New field name. Maximum 128 characters.

`entries[].overwrite_if_to_key_exists` (optional)

Whether to overwrite existing target field.

`when` (optional)

Processor-level conditional expression. When specified, the entire processor is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when` (optional)

Entry-level conditional expression. When specified, only this entry is skipped if the expression evaluates to false. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

`entries[].when_else` (optional)

Fallback entry that executes only when none of the other `when` conditions in the same processor matched. The expression value identifies which `when` conditions to consider. Maximum length is 256 characters. See the section called "Expression syntax for conditional processing".

select_entries processor

Selects only specified fields from events.

Configuration

Configure the select_entries processor with the following parameters:

```
processor:
  - select_entries:
      include_keys: ["timestamp", "level", "message"]
```

Parameters

`include_keys` (required)

Array of field names to keep. Maximum 50 keys, each up to 128 characters.

`when` (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

translate processor

Translates field values using lookup tables.

Configuration

Configure the translate processor with the following parameters:

```
processor:
  - translate:
      mappings:
        - source: "status_code"
          targets:
            - target: "status_text"
              map:
                "200": "OK"
                "404": "Not Found"
```

Parameters

`mappings` (required)

Array of translation configurations. Maximum 10 mappings.

`mappings[].source` (required)

Field to translate. Maximum 128 characters.

`mappings[].targets` (required)

Array of target configurations. Maximum 10 targets.

`mappings[].targets[].target` (required)

Target field name. Maximum 128 characters.

`mappings[].targets[].map` (required)

Translation mapping. Maximum 100 entries, each value up to 512 characters.

when (optional)

Conditional expression that determines whether this processor executes. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing”.

Filter processors

Filter processors let you selectively remove log entries from the pipeline based on conditions you define.

drop_events processor

Filters out unwanted log entries based on conditional expressions. Use this processor to reduce noise from third-party pipeline connectors and lower storage costs by removing log events that match specified conditions.

Configuration

Configure the drop_events processor with the following parameters:

```
processor:
  - drop_events:
      when: "log.level == 'DEBUG' or log.level == 'TRACE'"
```

Parameters

when (required)

Conditional expression that determines which log entries to drop. Log entries matching this expression are removed from the pipeline. Maximum length is 256 characters. See the section called “Expression syntax for conditional processing” for expression syntax.

handle_expression_failure (optional)

Behavior when the when expression evaluation fails. Allowed values: "skip" (default) keeps the event, or "apply" drops the event regardless of the failure.

Example Drop low-severity log entries

The following configuration drops all DEBUG and TRACE log entries, keeping only higher-severity events:

```
processor:
  - drop_events:
      when: "log.level in {'DEBUG', 'TRACE'}"
      handle_expression_failure: "skip"
```

Common processor use cases

Here are common scenarios and example configurations for combining processors.

Logs pipeline examples

Example Standardize log formats and add metadata

Parse JSON logs, standardize field names, and add environment information:

```
processor:
  - parse_json: {}
  - rename_keys:
      entries:
        - from_key: "timestamp"
          to_key: "@timestamp"
        - from_key: "log_level"
          to_key: "level"
  - add_entries:
      entries:
        - key: "environment"
          value: "production"
        - key: "application"
          value: "payment-service"
```

Example Clean and normalize field values

Standardize status codes and remove sensitive data:

```
processor:
  - uppercase_string:
      with_keys: ["status", "method"]
  - delete_entries:
      with_keys: ["credit_card", "password"]
  - substitute_string:
      entries:
        - source: "status"
```

```
          from: "SUCCESS"
          to: "OK"
```

Example Extract and transform specific fields

Extract user information and format for analysis:

```
processor:
  - extract_value:
      entries:
        - source: "user_agent"
          target: "browser"
          from: "(?<browser>Chrome|Firefox|Safari)"
          to: "${browser}"
  - lowercase_string:
      with_keys: ["browser"]
  - move_keys:
      entries:
        - from_key: "browser"
          to_key: "user_data.browser"
```

Example Conditional processing with entry-level conditions

Add different metadata based on log severity using entry-level when conditions:

```
processor:
  - add_entries:
      entries:
        - key: "alert_level"
          value: "critical"
          when: "log.level == 'ERROR'"
        - key: "alert_level"
          value: "info"
          when_else: "log.level == 'ERROR'"
```

Example Drop unwanted log entries

Filter out debug and trace log entries from a third-party source to reduce noise and storage costs:

```
processor:
  - drop_events:
      when: "log.level in {'DEBUG', 'TRACE'}"
```

```
      handle_expression_failure: "skip"
```

Example Processor-level conditional with delete_entries

Remove sensitive fields only when the environment is production:

```
processor:
  - delete_entries:
      with_keys: ["password", "api_key", "ssn"]
      when: "environment in {'prod', 'staging'}"
```

Metrics pipeline examples

The following examples show processor configurations for metrics pipelines. Metrics processors use OTTL path expressions to target attributes at different scopes.

Example Add business context to metrics

Add team ownership and environment tags to metric datapoints:

```
processor:
  - add_attributes:
      attributes:
        - key: resource.attributes["team"]
          value: "platform-engineering"
        - key: resource.attributes["cost_center"]
          value: "CC-1234"
```

Example Remove high-cardinality attributes

Strip attributes that drive up storage costs. Does not apply to cumulative metrics or vended metrics — if any metrics in the selection criteria have unsupported temporality, the pipeline emits an `UnsupportedTemporality` warning metric that you can monitor in the `AWS/Observability Admin` namespace:

```
processor:
  - delete_attributes:
      with_keys:
        - resource.attributes["host.id"]
        - datapoint.attributes["http.request.id"]
```

Example Standardize naming conventions

Rename metrics and attributes to align with OpenTelemetry semantic conventions. Does not apply to cumulative metrics or vended metrics:

```
processor:
  - rename_metrics:
      metrics:
        - from: "cpu_usage_percent"
          to: "system.cpu.utilization"
  - rename_attributes:
      attributes:
        - from_key: resource.attributes["hostname"]
          to_key: resource.attributes["host.name"]
```

Processor compatibility and restrictions

General processor rules

Maximum count

A pipeline can have at most 20 processors.

Parser placement

Parser processors (OCSF, CSV, Grok, and so on), if used, must be the first processor in a pipeline.

Unique processors

The following processors can appear only once per pipeline:

- `add_entries`
- `copy_values`

Processor Type	CloudWatch Logs Source	S3 Source	API-based Sources
OCSF	Must be first processor	Must be first processor	Must be first processor
parse_vpc	Must be first processor	Not applicable	Not applicable

Processor Type	CloudWatch Logs Source	S3 Source	API-based Sources
parse_route53	Must be first processor	Not applicable	Not applicable
parse_rds	Must be first processor	Not applicable	Not applicable
parse_json	Must be first processor	Must be first processor	Must be first processor
grok	Must be first processor	Must be first processor	Must be first processor
csv	Must be first processor	Not compatible	Not compatible
key_value	Must be first processor	Must be first processor	Must be first processor
add_entries	Must be first processor	Must be first processor	Must be first processor
copy_values	Must be first processor	Must be first processor	Must be first processor
String processors (lowercase, uppercase, trim)	Must be first processor	Must be first processor	Must be first processor
Field processors (move_keys, rename_keys)	Must be first processor	Must be first processor	Must be first processor
Data transformation (date, flatten)	Must be first processor	Must be first processor	Must be first processor

Compatibility definitions

Must be first processor

When used, must be the first processor in the pipeline configuration

Not compatible

Cannot be used with this source type

Not applicable

Processor is not relevant for this source type

Processor-specific restrictions

The following table describes processor restrictions by source type.

Processor	Source Type	Restrictions
OCSF	CloudWatch Logs with CloudTrail	• Only allowed when `data_source_name` is `aws_cloudtrail` • Must use CloudTrail-specific schema version • Cannot be combined with other processors
OCSF	API-based Sources	• Must use source-specific schema (for example, microsoft_office365_management_activity for Office 365) • Requires specific mapping version for each source type • Must be first processor in pipeline
parse_vpc	CloudWatch Logs	• Only valid for VPC Flow Logs • Must be first processor • Input must contain raw VPC Flow Log format
parse_route53	CloudWatch Logs	• Only valid for Route 53 Resolver Query Logs

Processor	Source Type	Restrictions
		• Must be first processor • Input must contain Route 53 Resolver query log format
parse_rds	CloudWatch Logs	• Only valid when `data_source_name` is `amazon_rds` • Must be first processor • Takes no parameters; the log type is inferred from the pipeline's `data_source_type`
add_entries	All Sources	• Maximum one instance per pipeline • Key names must be valid according to field naming rules
copy_values	All Sources	• Maximum one instance per pipeline • Source fields must exist in the event

Important

When using processors with restrictions:

- Always validate your pipeline configuration using the `ValidateTelemetryPipelineConfiguration` API before deployment
- Test the pipeline with sample data using the `TestTelemetryPipeline` API to make sure proper processing
- Monitor pipeline metrics after deployment to make sure events are being processed as expected

Metrics pipeline processors

Metrics pipeline processors transform OTel metric datapoints during ingestion. A metrics pipeline can have up to 20 processors, applied sequentially in the order they are defined.

Processors use OTTL path expressions to target attributes at different scopes:

Path	Scope	Example
`resource.attributes["key"]`	Resource-level	`resource.attributes["service.name"]`
`instrumentation_scope.attributes["key"]`	Scope attribute	`instrumentation_scope.attributes["library"]`
`datapoint.attributes["key"]`	Datapoint-level	`datapoint.attributes["status_code"]`
`attributes["key"]`	Short form for datapoint	`attributes["environment"]`

add_attributes processor

Adds or overwrites attributes on metric datapoints. Works with all temporality types.

OTTL equivalent: `set(attributes["key"], "value")`

Configuration

Configure the add_attributes processor with the following parameters:

```
processor:
  - add_attributes:
      attributes:
        - key: resource.attributes["team"]
          value: "payments"
        - key: attributes["environment"]
          value: "production"
        - key: instrumentation_scope.attributes["version"]
          value: "1.0"
          overwrite_if_key_exists: true
```

Parameters

`attributes` (required)

Array of attribute objects to add.

`attributes[].key` (required)

OTTL attribute path specifying where to add the attribute.

`attributes[].value` (required)

String value to assign.

`attributes[].overwrite_if_key_exists` (optional)

Boolean. When `true`, overwrites existing values. Defaults to `false`. When set to `true`, this operation is considered destructive and does not apply to cumulative metrics or vended metrics.

delete_attributes processor

Removes specific attributes from metric datapoints. **Does not apply to cumulative metrics or vended metrics.**

OTTL equivalent: `delete_key(attributes, "key")`

Configuration

Configure the delete_attributes processor with the following parameters:

```
processor:
  - delete_attributes:
      with_keys:
        - resource.attributes["obsolete"]
        - attributes["temp"]
        - datapoint.attributes["debug"]
```

Parameters

`with_keys` (required)

Array of OTTL attribute paths to remove.

rename_attributes processor

Renames attribute keys on metric datapoints. **Does not apply to cumulative metrics or vended metrics.**

OTTL equivalent: `set() + delete_key()`

Configuration

Configure the rename_attributes processor with the following parameters:

```
processor:
  - rename_attributes:
      attributes:
        - from_key: resource.attributes["old_key"]
          to_key: resource.attributes["new_key"]
          overwrite_if_to_key_exists: true
```

Parameters

`attributes` (required)

Array of rename operations.

`attributes[].from_key` (required)

Source OTTL attribute path.

`attributes[].to_key` (required)

Target OTTL attribute path.

`attributes[].overwrite_if_to_key_exists` (optional)

Boolean. When `true`, overwrites if target key exists. Defaults to `false`.

rename_metrics processor

Renames metric names. **Does not apply to cumulative metrics or vended metrics.**

OTTL equivalent: `set(name, "new.metric.name") where name == "old.metric.name"`

Configuration

Configure the rename_metrics processor with the following parameters:

```
processor:
  - rename_metrics:
      metrics:
        - from: "old.metric.name"
```

```
          to: "new.metric.name"
```

Parameters

`metrics` (required)

Array of rename operations.

`metrics[].from` (required)

Current metric name.

`metrics[].to` (required)

New metric name.

substitute_attribute_values processor

Maps attribute values by using a lookup table. **Does not apply to cumulative metrics or vended metrics.**

OTTL equivalent: `replace_match()`

Configuration

Configure the substitute_attribute_values processor with the following parameters:

```
processor:
  - substitute_attribute_values:
      attributes:
        - key: resource.attributes["region"]
          from: "us-east-1a"
          to: "us-east-1"
```

Parameters

`attributes` (required)

Array of substitution operations.

`attributes[].key` (required)

OTTL attribute path to match against.

`attributes[].from` (required)

Value to match.

`attributes[].to` (required)

Replacement value.

Full example

The following example uses all five processors in a single pipeline:

```
pipeline:
  source:
    cloudwatch_metrics:
      format: otlp
      selection_criteria:
        - match_all:
            - 'resource.attributes["service.name"] == "my-service"'
            - 'metric.name == "CPUUtilization"'
  processor:
    - add_attributes:
        attributes:
          - key: resource.attributes["env"]
            value: "prod"
            overwrite_if_key_exists: true
    - delete_attributes:
        with_keys:
          - resource.attributes["internal.trace"]
    - rename_attributes:
        attributes:
          - from_key: resource.attributes["old_key"]
            to_key: resource.attributes["new_key"]
    - rename_metrics:
        metrics:
          - from: "CPUUtilization"
            to: "system.cpu.utilization"
    - substitute_attribute_values:
        attributes:
          - key: resource.attributes["environment"]
            from: "dev"
            to: "development"
  sink:
```

```
    - cloudwatch_metrics: {}
```

Processor compatibility and restrictions

Maximum processors

A metrics pipeline can have at most 20 processors.

Destructive processors and temporality

Destructive processors (`delete_attributes`, `rename_attributes`, `rename_metrics`, `substitute_attribute_values`, and `add_attributes` with `overwrite_if_key_exists: true`) do not apply to cumulative metrics or vended metrics. These metrics are passed through unchanged.

Vended metrics protection

Destructive processors cannot modify metrics where `instrumentation_scope.name` starts with `cloudwatch.aws/`. These metrics are passed through unchanged.

Expression syntax for conditional processing

CloudWatch pipelines processors and CloudWatch Logs sink routes that support conditional processing accept a `when` parameter containing an expression. When the expression evaluates to true, the processor or entry executes, or the event is sent to that sink route. Expressions use dot notation (`.`) for nested field access. For example, `user.role` accesses the `role` field inside the `user` object. For details about conditional sink routes, see the section called "Route logs to multiple log groups". For details about processors that support conditional processing, see the section called "Processors". For processor configuration examples, see the section called "Common processor use cases".

> **Note**
>
> Conditional processing applies to log pipelines only. Metrics pipelines do not support the `when` parameter.

Processor-level and entry-level conditions

There are two levels at which you can apply a `when` condition, depending on the processor.

Processor-level `when` (outer level)

A `when` placed at the top level of the processor configuration. If the expression evaluates to false, the entire processor is skipped and no operations within it execute. All processors that support conditional processing support this level.

Example Processor-level condition — skip entire processor

The following `delete_entries` processor only runs when the environment is production or staging. If the condition is false, none of the keys are deleted.

```
processor:
  - delete_entries:
      with_keys: ["password", "api_key", "ssn"]
      when: "environment in {'prod', 'staging'}"
```

Entry-level `when` (within each entry)

A `when` placed inside an individual entry in the `entries` array. Each entry is evaluated independently — if the expression is false, only that specific entry is skipped while other entries in the same processor still execute. Only processors with an `entries` array support this level (such as `add_entries`, `copy_values`, `rename_keys`, `move_keys`, `extract_value`, and `substitute_string`).

Example Entry-level condition — skip individual entries

The following `add_entries` processor adds different keys depending on each entry's condition. The first entry only adds `severity` when the log level is ERROR. The second entry always adds `processed` because it has no condition.

```
processor:
  - add_entries:
      entries:
        - key: "severity"
          value: "high"
          when: "log.level == 'ERROR'"
        - key: "processed"
          value: "true"
```

Processors that support both levels can use them together. When both are specified, the processor-level condition is evaluated first. If it is false, the entire processor is skipped and no entry-level conditions are evaluated.

Example Both levels combined

The processor-level `when` ensures the entire processor only runs for production traffic. Within that, each entry has its own condition to control which key is added.

```
processor:
  - add_entries:
      when: "environment == 'prod'"
      entries:
        - key: "alert_level"
          value: "critical"
          when: "log.level == 'ERROR'"
        - key: "alert_level"
          value: "warning"
          when: "log.level == 'WARN'"
```

For a table showing which processors support which level, see the the section called “Conditional processing support” section below.

Fallback with `when_else`

Processors that support entry-level conditions also support `when_else`. An entry with `when_else` acts as a fallback — it executes only when none of the other `when` conditions in the same processor matched. The expression value provided to `when_else` identifies which set of `when` conditions to consider, but the entry itself runs based solely on whether those conditions all evaluated to false. There is no explicit negation check — the entry simply runs when no other `when` matched.

Example Fallback entry with when_else

The first entry runs when the log level is ERROR. The second entry uses `when_else` and runs only when the first entry's `when` condition did not match (that is, the log level is anything other than ERROR).

```
processor:
  - add_entries:
      entries:
        - key: "alert_level"
          value: "critical"
```

```
          when: "log.level == 'ERROR'"
        - key: "alert_level"
          value: "info"
          when_else: "log.level == 'ERROR'"
```

Conditional processing support

The following table shows which processors support conditional processing and at what level.

Processor	Conditional support	Level
add_entries	Yes	Processor and entry
copy_values	Yes	Processor and entry
delete_entries	Yes	Processor
move_keys	Yes	Processor and entry
flatten	Yes	Processor
lowercase_string	Yes	Processor
uppercase_string	Yes	Processor
trim_string	Yes	Processor
substitute_string	Yes	Processor and entry
truncate	Yes	Processor
extract_value	Yes	Processor and entry
convert_entry_type	Yes	Processor
date	Yes	Processor
dissect	Yes	Processor
list_to_map	Yes	Processor

Processor	Conditional support	Level
rename_keys	Yes	Processor and entry
select_entries	Yes	Processor
translate	Yes	Processor
grok	Yes	Processor
drop_events	Yes	Processor (required)
OCSF, CSV, JSON, KeyValue, WAF, Postgres, CloudFront, VPC, Route53	No	—

Operators

The following table lists the supported operators.

Category	Operators	Example
Relational	<, <=, >, >=	`status_code >= 200 and status_code < 300`
Equality	==, !=	`log.level == "ERROR"`
Conditional	and, or, not	`log.level == "ERROR" or log.level == "FATAL"`
Arithmetic	+, -, *, /	`response_time * 1000 > 5000`
Set membership	in, not in	`environment in {"prod", "staging", "preprod"}`
Regex matching	=~, !~	`message =~ "^ERROR.*timeout"`

Functions

`length(value)`

Returns the length of a string or array. Example: `length(message) > 100`

`contains(value, search)`

Checks whether a string contains a substring or an array contains an element. Example: `contains(message, "error")`

`startsWith(field, prefix)`

Checks whether a string starts with a specified prefix. Example: `startsWith(message, "ERROR")`

Expression examples

```
log.level == "ERROR"
status_code >= 200 and status_code < 300
environment in {"prod", "staging", "preprod"}
message =~ "^ERROR.*timeout"
user.role == "admin" and user.permissions.write == true
length(message) > 100 and contains(message, "error")
(log.level == "ERROR" or log.level == "FATAL") and environment == "prod"
```

Limitations

- Expression maximum length is 256 characters.
- Parser processors (except Grok) do not support conditional processing. This includes JSON, CSV, KeyValue, WAF, Postgres, CloudFront, VPC, Route53, and OCSF parsers.
- If the Grok processor is used as the parser (first processor) in a pipeline and its `when` condition evaluates to false, the entire pipeline does not execute for that log event.

Sinks

Sinks define the destination where processed data is sent. Logs pipelines use the `cloudwatch_logs` sink and can route events among multiple log groups. Metrics pipelines use exactly one `cloudwatch_metrics` sink.

CloudWatch Logs sink (`cloudwatch_logs`)

For logs pipelines, the `cloudwatch_logs` sink sends processed log events to a CloudWatch Logs log group.

Source Type	Log group configuration	Behavior
CloudWatch Logs	Use `@original` for a pipeline with one sink, or configure conditional sinks for routing	Events are returned to their original log group or routed to the selected destination log group
S3	Custom log group path	Events are sent to the specified log group
Third-party APIs	Custom log group path	Events are sent to the specified log group

Configuration

Configure the sink with the following parameters:

Example Non-CloudWatch Logs source configuration

```
sink:
  cloudwatch_logs:
    log_group: "/aws/my-application/logs"
```

Example CloudWatch Logs source configuration

```
sink:
  cloudwatch_logs:
    log_group: "@original"
```

Parameters

`log_group` (required)

The name of the CloudWatch Logs log group where processed events will be sent. For pipelines with non-`cloudwatch_logs` sources, this must be an existing log group name. For a pipeline

that has one `cloudwatch_logs` sink and uses a `cloudwatch_logs` source, the value must be `@original`. A pipeline that uses conditional routing can use `@original` or another destination log group in any route.

`when` (optional)

An expression that selects events for this route. Conditions are evaluated in sink order, and the first matching route is used. The expression can contain up to 4,096 characters. For syntax, see Expression syntax for conditional processing.

`is_default` (optional)

Set to `true` on exactly one sink when the pipeline has multiple sinks. This route receives events that don't match an earlier `when` expression. Place the default route last.

`include_original` (optional)

When present, stores a copy of each raw log event before any transformation takes place in the `@original_message` system field of the event. This preserves the original data for audit or compliance purposes. Specify as an empty object (`{}`). Available only for pipelines with `cloudwatch_logs` sources. At least one processor must be configured when this option is enabled.

Example CloudWatch Logs sink with original log preservation

```
sink:
  - cloudwatch_logs:
      log_group: "@original"
      include_original: {}
```

Route logs to multiple log groups

A logs pipeline can contain up to 10 `cloudwatch_logs` sinks, including the default route. Define a `when` expression on each conditional route and set `is_default: true` on exactly one fallback route. Each event is delivered to one destination only.

Example Route by log level and return unmatched events to the source log group

```
sink:
  - cloudwatch_logs:
      log_group: "/app/errors"
      when: 'level == "ERROR"'
```

```
- cloudwatch_logs:
    log_group: "/app/warnings"
    when: 'level == "WARN"'
- cloudwatch_logs:
    log_group: "@original"
    is_default: true
```

For an S3 source, a destination log group can contain one template variable. Use `{s3.path.N}` to select the Nth segment of the S3 object key, starting at 1, or use a field beneath `{@pipeline_metadata.source.s3}`, such as `{@pipeline_metadata.source.s3.bucket}`. For example, `/customers/{s3.path.1}` routes an object with the key `customer-a/audit.json` to `/customers/customer-a`.

A resolved template value must contain 1–64 letters, numbers, periods, underscores, or hyphens. Each template route can resolve to at most five distinct destination log groups for one input batch. Events that exceed this limit, or whose template can't be resolved, use the configured default route. Static routes don't count toward this template limit.

Requirements and limitations

Log group existence

If created using the AWS Management Console, CloudWatch will attempt to create the specified log group and appropriate resource policy if it does not exist when using a non-CloudWatch logs source. Otherwise, the specified log group must exist before creating the pipeline. For conditional routing, a physical default-route log group must exist before you create the pipeline. Conditional destination log groups are created when first used. The `@original` alias resolves to the source log group and is not created.

Event size

Each log event cannot exceed 256 KB in size after processing.

Log group retention

The pipeline uses the retention settings configured on the destination log group.

Log group resource policy

CloudWatch Logs resource policies are required for the direct destination of S3-based and third-party API pipelines. For a `cloudwatch_logs` source, the `@original` destination does not require a resource policy. Additional named conditional and templatized destinations

use the pipeline source role. When you use the AWS Management Console to configure the pipeline, CloudWatch will attempt to add the direct-destination resource policy if needed. If you create the pipeline using the AWS CLI or an API, you must create the policy manually. For more information, see Resource policies and Permissions for routed destinations.

Cross-Region support

The destination log group must be in the same Region as the pipeline.

Important

For pipelines using the `cloudwatch_logs` source type:

- A pipeline with one sink must use `@original` as the log group value. Conditional routing can use other destination log groups.
- Use `@original` in any route that should return matching events to their source log group.
- The original log group must exist throughout the pipeline's lifecycle.
- Pipelines with processors mutate the log events in the original CloudWatch log group they are intercepted from for logs from AWS services. To preserve the original data, enable `include_original` in the sink configuration. The original log copy is stored in the `@original_message` system field of the event.

Note

Log events are subject to CloudWatch Logs quotas and limitations.

CloudWatch Metrics sink (`cloudwatch_metrics`)

For metrics pipelines, the `cloudwatch_metrics` sink stores processed metrics in CloudWatch. Processed metrics are queryable through PromQL in Query Studio and fully compatible with CloudWatch Alarms, Anomaly Detection, and Dashboards.

Configuration

The `cloudwatch_metrics` sink has no parameters:

```
sink:
  - cloudwatch_metrics: {}
```

CloudWatch pipelines IAM policies and permissions

This section provides IAM requirements for CloudWatch pipelines. Permissions vary based on your data source and integration method.

The following table helps you identify which IAM sections apply to your use case.

Use case	Integration method	Source type in pipeline configuration	IAM sections you need
Third-party integrations (API Pull)	Pipeline pulls from vendor API using stored credentials	`microsoft_office365` , `okta_sso`, `palo_alto_ngfw` , and so on.	the section called “API caller permissions” + the section called “Third-party sources (API Pull)” + the section called “Resource policies”
Third-party integrations (S3 delivery)	Vendor delivers files to your S3 bucket	`s3`	the section called “API caller permissions” + the section called “Third-party sources (S3 delivery)” + the section called “Resource policies”
Custom data from S3	Your applications write to S3, pipeline reads from bucket	`s3`	the section called “API caller permissions” + the section called “Custom data from S3” + the section called “Resource policies”
Custom and Syslog data from CloudWatch Logs	Your applications or the CloudWatch Logs Syslog service deliver logs to a log group	`cloudwatch_logs`	the section called “API caller permissions” + the section called “Custom and Syslog data from CloudWatch Logs”

Use case	Integration method	Source type in pipeline configuration	IAM sections you need
Vended AWS service logs	AWS services deliver logs to CloudWatch Logs (VPC Flow Logs, Route 53)	`cloudwatch_logs`	the section called "API caller permissions" + the section called "Vended AWS service logs"
CloudWatch Metrics (OTel)	AWS services emit metrics through OTLP	`cloudwatch_metrics`	the section called "Pipeline rule permissions for CloudWatch Metrics sources"

> **Note**
>
> S3-based and third-party API sources require a resource policy for their direct CloudWatch Logs destination. For CloudWatch Logs sources, the `@original` destination does not require a resource policy. Named conditional and templatized destinations use the pipeline source role. For more information, see the section called "Permissions for routed destinations".

API caller permissions

The IAM principal that calls `CreateTelemetryPipeline` needs `iam:PassRole` permission for any roles referenced in the pipeline configuration.

Example PassRole policy template

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PassRoleForPipelineSource",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/your-source-role",
```

```
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "service-principal"
                    ],
                    "iam:AssociatedResourceARN": [
                        "arn:aws:observabilityadmin:us-east-1:111122223333:telemetry-
pipeline/*"
                    ]
                }
            }
        }
    ]
}
```

Replace *service-principal* with the value from the following table based on your use case.

Use case	Service principal value
Third-party (API Pull)	telemetry-pipelines.observabilityadmin.amazonaws.com
Third-party (S3 delivery)	telemetry-pipelines.observabilityadmin.amazonaws.com
Custom data from S3	telemetry-pipelines.observabilityadmin.amazonaws.com
Custom data from CloudWatch Logs	logs.amazonaws.com
Vended AWS service logs	logs.amazonaws.com

Note

The Condition block is recommended but optional. Without it, the role can be passed to any service.
For third-party pipelines that use a lookup processor or conditional routing, also allow logs.amazonaws.com in iam:PassedToService.

Pipeline rule permissions (CloudWatch Logs sources only)

When using `cloudwatch_logs` as a source, the API caller also needs permissions for pipeline rule operations. The `logs:PutPipelineRule` permission is required for `CreateTelemetryPipeline` and `UpdateTelemetryPipeline` operations. The `logs:DeletePipelineRule` permission is required for `DeleteTelemetryPipeline` operations.

Example IAM policy for CloudWatch Logs pipeline rules

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PipelineRuleForCloudWatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutPipelineRule",
                "logs:DeletePipelineRule"
            ],
            "Resource": "*"
        }
    ]
}
```

Pipeline rule permissions for CloudWatch Metrics sources

When you use `cloudwatch_metrics` as a source, you need permissions for pipeline rule operations. To create or update a pipeline, grant the `cloudwatch:PutPipelineRule` permission. To delete a pipeline, grant the `cloudwatch:DeletePipelineRule` permission. Metrics pipelines do not require `iam:PassRole` or CloudWatch Logs resource policies. You can scope these actions down to the `dataset/default` resource.

Example IAM policy for CloudWatch metrics pipeline rules

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PipelineRuleForCloudWatchMetrics",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutPipelineRule",
```

```
                "cloudwatch:DeletePipelineRule"
            ],
            "Resource": "arn:aws:cloudwatch:us-east-1:111122223333:dataset/default"
        }
    ]
}
```

Reducing scope with condition keys

Source role policies

Each pipeline requires a dedicated IAM role that the service assumes to read your data. The following subsections provide the complete policies (permission and trust) for each use case.

Third-party sources (API Pull)

This section applies to Microsoft Office 365, Microsoft Entra ID, Okta SSO, Palo Alto NGFW, and other vendor API integrations that store credentials in AWS Secrets Manager.

Permission policy

The following policy allows the role to retrieve your stored API credentials.

Example IAM policy for Secrets Manager sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "secretsmanageraccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:your-
secret-name*"
        },
        {
            "Sid": "kmsaccess",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/your-key-id"
        }
```

```
    ]
}
```

> **Note**
>
> The `kms:Decrypt` statement is only required if your secret in Secrets Manager is encrypted with a customer-managed KMS key.

Trust policy

Example Trust policy for API Pull sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Complete IAM setup for API Pull pipelines

The following example shows all the IAM policies needed to create a third-party API Pull pipeline end to end.

Caller identity policy — attach to the principal calling `CreateTelemetryPipeline`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateApiPullPipeline",
            "Effect": "Allow",
            "Action": "observabilityadmin:CreateTelemetryPipeline",
            "Resource": "*"
```

```
        },
        {
            "Sid": "PassRoleForApiPullPipeline",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/your-source-role",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "telemetry-pipelines.observabilityadmin.amazonaws.com"
                    ],
                    "iam:AssociatedResourceARN": [
                        "arn:aws:observabilityadmin:us-east-1:111122223333:telemetry-
pipeline/*"
                    ]
                }
            }
        }
    ]
}
```

Source role permission policy — attach to the role the pipeline assumes:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:111122223333:secret:your-
secret-name*"
        }
    ]
}
```

Source role trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Note

After creating the pipeline, you must also create a resource policy within 5 minutes. See the section called “Resource policies”.

Note

For production use, scope down `iam:PassRole` by using the condition keys shown in the section called “API caller permissions”. If your secret uses a customer-managed KMS key, add `kms:Decrypt` to the source role permission policy.

Third-party sources (S3 delivery)

This section applies to any third-party vendor that delivers log files to your S3 bucket (for example, CrowdStrike Falcon, Wiz, or Cisco Umbrella).

Permission policy

The following policy allows the role to read objects from S3 and consume SQS notifications.

Example IAM policy for S3 sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "s3access",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::your-bucket-name/*"
        },
        {
            "Sid": "sqsaccess",
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:ChangeMessageVisibility"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111122223333:your-queue-name"
        },
        {
            "Sid": "kmsaccess",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/your-key-id"
        }
    ]
}
```

> **Note**
>
> The `kms:Decrypt` statement is only required if your S3 bucket or SQS queue uses a customer-managed KMS key for encryption.

Trust policy

Example Trust policy for S3 delivery sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
```

```
        }
    ]
}
```

Complete IAM setup for S3 delivery pipelines

The following example shows all the IAM policies needed to create an S3 delivery pipeline end to end.

Caller identity policy — attach to the principal calling `CreateTelemetryPipeline`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateS3Pipeline",
            "Effect": "Allow",
            "Action": "observabilityadmin:CreateTelemetryPipeline",
            "Resource": "*"
        },
        {
            "Sid": "PassRoleForS3Pipeline",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/your-source-role",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "telemetry-pipelines.observabilityadmin.amazonaws.com"
                    ],
                    "iam:AssociatedResourceARN": [
                        "arn:aws:observabilityadmin:us-east-1:111122223333:telemetry-
pipeline/*"
                    ]
                }
            }
        }
    ]
}
```

Source role permission policy — attach to the role the pipeline assumes:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::your-bucket-name/*"
        },
        {
            "Sid": "SqsAccess",
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:ChangeMessageVisibility"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111122223333:your-queue-name"
        }
    ]
}
```

Source role trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> After creating the pipeline, you must also create a resource policy within 5 minutes. See the section called “Resource policies”.

> **Note**
>
> For production use, scope down `iam:PassRole` by using the condition keys shown in the section called "API caller permissions". If your S3 bucket or SQS queue uses a customer-managed KMS key, add `kms:Decrypt` to the source role permission policy.

Custom data from S3

This section applies to your own applications or infrastructure writing log files to an S3 bucket.

The IAM setup is identical to the section called "Third-party sources (S3 delivery)". Use the same permission policy and trust policy. The pipeline reads from your bucket through SQS event notifications regardless of who wrote the data.

Custom and Syslog data from CloudWatch Logs

This section applies to your own applications logging to a CloudWatch Logs log group (for example, Lambda functions, ECS containers, or custom EC2 applications) and to messages delivered through the CloudWatch Logs Syslog service.

Permission policy

The following policy allows the role to process logs from your specified log groups.

Example IAM policy for CloudWatch Logs sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "logsprocessingaccess",
            "Effect": "Allow",
            "Action": [
                "logs:processWithPipeline"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-01",
                "arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-02"
            ]
        }
```

```
    ]
}
```

You can scope down this permission by using the `logs:data_source_name` and `logs:data_source_type` condition keys to restrict which pipeline sources can invoke transformations. The `logs:data_source_name` value corresponds to the `data_source_name` in your pipeline configuration, and `logs:data_source_type` corresponds to the `data_source_type` in your pipeline configuration.

Example Permission policy for CloudWatch Logs sources (scoped down with condition keys)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowProcessWithPipelineScopedDown",
            "Effect": "Allow",
            "Action": "logs:ProcessWithPipeline",
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-
name",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "111122223333",
                    "logs:data_source_name": "your-source-name",
                    "logs:data_source_type": "your-source-type"
                }
            }
        }
    ]
}
```

> **Note**
>
> The IAM role for CloudWatch Logs sources requires both the trust policy (to allow `logs.amazonaws.com` to assume the role) and the permission policy (to grant `logs:ProcessWithPipeline`). Without both policies, CloudWatch pipelines cannot transform log events during ingestion.

Trust policy

Example Trust policy for CloudWatch Logs sources

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> No resource policy is needed for CloudWatch Logs source pipelines.

> **Note**
>
> CloudWatch Logs Syslog ingestion requires a separate resource policy to deliver messages to the source log group. For more information, see Setting up syslog ingestion.

Complete IAM setup for CloudWatch Logs pipelines

The following example shows all the IAM policies needed to create a CloudWatch Logs pipeline end to end.

Caller identity policy — attach to the principal calling `CreateTelemetryPipeline`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateLogsPipeline",
            "Effect": "Allow",
            "Action": [
```

```
                "observabilityadmin:CreateTelemetryPipeline",
                "logs:PutPipelineRule",
                "logs:DeletePipelineRule"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassRoleForLogsPipeline",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/your-source-role",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "logs.amazonaws.com"
                    ],
                    "iam:AssociatedResourceARN": [
                        "arn:aws:observabilityadmin:us-east-1:111122223333:telemetry-
pipeline/*"
                    ]
                }
            }
        }
    ]
}
```

Source role permission policy — attach to the role the pipeline assumes:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LogsProcessing",
            "Effect": "Allow",
            "Action": [
                "logs:processWithPipeline"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:your-log-group"
            ]
        }
    ]
}
```

Source role trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

> **Note**
>
> For production use, scope down `iam:PassRole` by using the condition keys shown in the section called "API caller permissions".

Vended AWS service logs

This section applies to AWS service logs delivered to CloudWatch Logs, such as VPC Flow Logs, Route 53 query logs, and other AWS vended log types.

The IAM setup is identical to the section called "Custom and Syslog data from CloudWatch Logs". Use the same permission policy (`logs:processWithPipeline` scoped to the log group) and the same trust policy (`logs.amazonaws.com`).

Because this uses the `cloudwatch_logs` source type, the caller also needs `logs:PutPipelineRule` and `logs:DeletePipelineRule` permissions. See the section called "Pipeline rule permissions (CloudWatch Logs sources only)".

Resource policies

CloudWatch Logs resource policies are required for the direct CloudWatch Logs destination of an S3-based or third-party API pipeline. The direct destination is the single sink, or the named default route when conditional routing is configured.

For a CloudWatch Logs source, the `@original` destination does not require a resource policy. Named conditional and templatized destinations use the pipeline source role instead of the CloudWatch pipelines service principal. For more information, see the section called “Permissions for routed destinations”.

After calling `CreateTelemetryPipeline`, you receive a pipeline ARN. You must then call `logs:PutResourcePolicy` to allow the CloudWatch pipelines service principal to write to the direct destination log group. The principal that creates the policy needs `logs:PutResourcePolicy`. To inspect or update an existing policy, the principal also needs `logs:DescribeResourcePolicies`.

Use a resource-scoped policy for the destination log group. Resource-scoped policies do not count toward the quota of 10 account-level resource policies per account per Region. The destination log group must exist before you create its resource-scoped policy.

> **Timing constraint**
>
> You have less than 5 minutes after receiving the pipeline ARN to create this resource policy. If the pipeline becomes active before the policy is in place, data will be dropped.

Example Resource policy for a direct destination

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name:*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:observabilityadmin:us-east-1:111122223333:telemetry-pipeline/your-pipeline-id"
```

```
                }
            }
        }
    ]
}
```

Permissions for routed destinations

Additional conditional and templatized destinations are authorized by identity-based permissions on the IAM role in the pipeline source configuration, not by an additional CloudWatch Logs resource policy.

The pipeline source role must allow `logs:CreateLogGroup` for log groups that the pipeline creates automatically. It must also allow `logs:CreateLogStream` and `logs:PutLogEvents` for every named routed destination. A resource policy does not grant `logs:CreateLogGroup`.

Example Permission policy for routed destinations

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateRoutedLogGroups",
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "*"
        },
        {
            "Sid": "WriteRoutedLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:your-region:your-account-id:log-group:your-routed-log-group-prefix*:*"
        }
    ]
}
```

Scope the write statement to the exact destination log groups or the narrowest prefix that contains them. Include `logs:CreateLogGroup` only when the pipeline must create conditional or templatized destination log groups.

The role trust policy must allow each service that assumes the role:

- For S3 and third-party API sources, allow `telemetry-pipelines.observabilityadmin.amazonaws.com`. When the pipeline uses conditional routing, also allow `logs.amazonaws.com`.
- For CloudWatch Logs sources, use `logs.amazonaws.com`.

Managing resource policies

Use the AWS CLI to create or update CloudWatch Logs resource policies for CloudWatch pipelines.

To check for existing policies

```
aws logs describe-resource-policies \
    --policy-scope RESOURCE \
    --resource-arn arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name
```

To create a new policy

```
aws logs put-resource-policy \
    --region us-east-1 \
    --resource-arn arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name \
    --policy-document file://policy.json
```

To merge with an existing policy

Each log group can have one resource-scoped policy. If a policy already exists, add the new statement to the existing `Statement` array, then call `put-resource-policy` with the merged file and the current revision ID.

1. Retrieve the existing policy:

```
aws logs describe-resource-policies \
    --policy-scope RESOURCE \
    --resource-arn arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name
```

2. Add the new statement to the existing `Statement` array:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "existing-service.amazonaws.com"
            },
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-
group-name:*"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-
group-name:*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:observabilityadmin:us-
east-1:111122223333:telemetry-pipeline/your-pipeline-id"
                }
            }
        }
    ]
}
```

3. Update the policy:

```
aws logs put-resource-policy \
    --region us-east-1 \
    --resource-arn arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name
 \
```

```
    --expected-revision-id your-revision-id \
    --policy-document file://existing-policy.json
```

Confirm the policy was created or updated successfully:

```
aws logs describe-resource-policies \
    --policy-scope RESOURCE \
    --resource-arn arn:aws:logs:us-east-1:111122223333:log-group:your-log-group-name
```

Replace the following values in the policy:

- The AWS Region (shown as `us-east-1`) – Replace with your own Region
- The 12-digit AWS account ID (shown as 111122223333) – Replace with your own account ID
- *your-log-group-name* – Your CloudWatch Logs log group name
- *your-pipeline-id* – Your telemetry pipeline ID (returned by `CreateTelemetryPipeline`)
- *your-revision-id* – The revision ID returned by `describe-resource-policies` for an existing resource-scoped policy

Pipeline condition keys

CloudWatch pipelines supports IAM condition keys that let you restrict who can create pipelines based on the source name and type. Use these condition keys to enforce governance policies across your organization.

Available condition keys

`observabilityadmin:SourceName`

Restricts pipeline creation to specific source names. Applies to logs pipelines only.

`observabilityadmin:SourceType`

Restricts pipeline creation to specific source types.

Use these condition keys in identity policies to control which pipelines a principal can create.

`observabilityadmin:SourceType`

Restricts pipeline creation to specific source types. Supported values include `cloudwatch_logs`, `s3`, `microsoft_office365`, `okta_sso`, and `palo_alto_ngfw`.

Example Restrict pipeline creation by source type

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPipelineCreationForSpecificSourceType",
            "Effect": "Allow",
            "Action": "observabilityadmin:CreateTelemetryPipeline",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "observabilityadmin:SourceType": "cloudwatch_logs"
                }
            }
        }
    ]
}
```

Source role trust policy conditions

Use these condition keys in the trust policy of your source role to restrict which account can assume the role. This prevents confused deputy attacks where the service might act on behalf of a different account.

`aws:SourceAccount`

Restricts role assumption to requests originating from a specific AWS account.

`aws:SourceArn`

Restricts role assumption to requests originating from a specific resource ARN (for example, a log group).

Example Trust policy with SourceAccount condition (CloudWatch Logs sources)

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:logs:us-east-1:111122223333:log-group:*"
                }
            }
        }
    ]
}
```

AI-assisted processor configuration permissions

To use AI-assisted processor configuration in the CloudWatch pipelines console, the IAM principal must have the `logs:GeneratePipeline` permission. This permission authorizes the generation of processor configurations from natural language descriptions.

Example IAM policy for AI-assisted processor configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGeneratePipeline",
            "Effect": "Allow",
            "Action": "logs:GeneratePipeline",
            "Resource": "*"
        }
    ]
}
```

Configuring lookup processor permissions

If your pipeline configuration includes a Lookup processor, the source role needs to have the `logs:GetLookupTable` permission on the lookup table ARN. The role must also be assumable by `logs.amazonaws.com`.

Add the following statement to the source role's permissions policy, for all source types.

Example Permissions policy statement for lookup table access

```
{
    "Sid": "LookupTableAccess",
    "Effect": "Allow",
    "Action": "logs:GetLookupTable",
    "Resource": "arn:aws:logs:us-east-1:111122223333:lookup-table:your-lookup-table-
name"
}
```

Replace the following values in the policy:

- The AWS Region (shown as `us-east-1`) – Replace with your own Region
- The 12-digit AWS account ID (shown as 111122223333) – Replace with your own account ID
- *`your-lookup-table-name`* – The name of your lookup table

The source role trust policy requirement differs by source type.

Custom data from CloudWatch Logs and vended AWS service logs

For the section called "Custom and Syslog data from CloudWatch Logs" and the section called "Vended AWS service logs", the role already trusts `logs.amazonaws.com`, so no trust policy change is required. The following trust policy shows the existing configuration.

Example Trust policy for CloudWatch Logs sources (no change required)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Third-party sources

For the section called “Third-party sources (API Pull)” and the section called “Third-party sources (S3 delivery)”, add a second statement to the trust policy so that `logs.amazonaws.com` can assume the role in addition to the existing telemetry service principal. The following trust policy shows both statements.

Example Trust policy for third-party sources (add logs.amazonaws.com)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "telemetry-pipelines.observabilityadmin.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Note

If your caller identity policy scopes `iam:PassRole` with the `iam:PassedToService` condition, list `logs.amazonaws.com` alongside `telemetry-pipelines.observabilityadmin.amazonaws.com`. An unconditioned `iam:PassRole` grant already covers both. For more information, see the section called “API caller permissions”.

CloudWatch pipelines extensions

CloudWatch pipelines extensions provide additional functionality to the pipeline. You can use the AWS Secrets Manager integration for credential management.

AWS Secrets Manager extension

Configures access to AWS Secrets Manager for retrieving credentials and sensitive configuration values. This extension is only supported for third-party sources that require authentication credentials.

Configuration

Configure the AWS Secrets Manager extension with the following parameters:

```
extension:
  aws:
    secrets:
      <secret-name>:
        secret_id: "<secret arn>"
        region: "<secret region>"
        sts_role_arn: "arn:aws:iam::123456789012:role/Example-Role"
        refresh_interval: PT1H
        disable_refresh: false
```

Parameters

`aws.secrets.<secret-name>.secret_id` (required)

The ARN of the AWS Secrets Manager secret containing the credentials.

`aws.secrets.<secret-name>.region` (required)

The AWS region where the secret is stored.

`aws.secrets.<secret-name>.sts_role_arn` (required)

The ARN of the IAM role to assume for accessing the AWS Secrets Manager secret.

`aws.secrets.<secret-name>.refresh_interval` (optional)

How often to refresh the secret from AWS Secrets Manager. Uses ISO 8601 duration format. Defaults to PT1H (1 hour).

`aws.secrets.<secret-name>.disable_refresh` (optional)

Whether to disable automatic secret refresh. Defaults to false.

Secret reference syntax

Reference secrets in your pipeline configuration using the following syntax:

```
${{aws_secrets:<secret-name>:<key>}}
```

For example, to reference a client ID and secret:

```
source:
  microsoft_office365:
    authentication:
      oauth2:
        client_id: "${{aws_secrets:office365-creds:client_id}}"
        client_secret: "${{aws_secrets:office365-creds:client_secret}}"
```

Requirements and limitations

Secret format

Secrets must be stored as JSON key-value pairs in AWS Secrets Manager.

Cross-Region access

Secrets can be accessed from any Region where AWS Secrets Manager is available.

Refresh interval limits

Minimum refresh interval is 5 minutes (PT5M). Maximum is 24 hours (PT24H).

Maximum secrets

A pipeline can reference up to 10 different secrets.

Important

Consider the following when using secrets:

- Make sure the IAM role has appropriate permissions to access the secrets
- Monitor secret access using AWS CloudTrail
- Use separate secrets for different environments (development, production)

Monitoring pipelines

CloudWatch pipelines publishes metrics to Amazon CloudWatch in the `AWS/Observability Admin` namespace. You can use these metrics to monitor your pipelines' health, performance, and data flow.

Available metrics

The following tables list the available metrics for CloudWatch pipelines.

Note

Pipelines metrics are only emitted when the value is non-zero.

Pipeline health metrics

The following metrics are emitted for all pipeline types (logs and metrics):

Metric	Description	Dimension	Unit
`PipelineErrors`	Aggregate count of errors in pipeline	PipelineName	Count

Metric	Description	Dimension	Unit
PipelineErrorsByErrorType	Detailed error counts by type	PipelineName, ErrorType	Count
PipelineWarnings	Number of warnings encountered	PipelineName	Count
PipelineWarningsByWarningType	Detailed warnings by type	PipelineName, WarningType	Count

> **Note**
>
> For logs pipelines, PipelineErrorsByErrorType includes additional dimensions (ErrorSource, ErrorComponent) and PipelineWarningsByWarningType includes additional dimensions (WarningSource, WarningComponent).

Logs pipeline metrics

The following metrics are emitted for logs pipelines:

Metric	Description	Dimension	Unit
PipelineBytesIn	Volume of log records going into pipeline in uncompressed bytes	PipelineName	Bytes
PipelineBytesInByDataSource	Volume of incoming data with source/type breakdown	PipelineName, DataSource, DataType	Bytes

Metric	Description	Dimension	Unit
PipelineBytesOut	Volume of data routed to destination	PipelineName	Bytes
PipelineBytesOutByDataSource	Volume of outgoing data with source/type breakdown	PipelineName, DataSource, DataType	Bytes
PipelineRecordsIn	Number of records entering the pipeline	PipelineName	Count
PipelineRecordsInByDataSource	Number of incoming records with source/type breakdown	PipelineName, DataSource, DataType	Count
PipelineRecordsOut	Number of records exiting the pipeline	PipelineName	Count
PipelineRecordsOutByDataSource	Number of outgoing records with source/type breakdown	PipelineName, DataSource, DataType	Count
PipelineRecordsUnprocessed	Number of records that couldn't be processed	PipelineName	Count
PipelineRecordsUnprocessedByDataSource	Number of unprocessed records with source/type breakdown	PipelineName, DataSource, DataType	Count

Metrics pipeline metrics

The following metrics are emitted for metrics pipelines:

Metric	Description	Dimension	Unit
`PipelineDataPointsIn`	Number of metric datapoints entering the pipeline (matched selection criteria)	PipelineName, dataset	Count
`PipelineDataPointsOut`	Number of metric datapoints exiting the pipeline after processing	PipelineName, dataset	Count
`PipelineDataPointsUnprocessed`	Number of metric datapoints that matched selection criteria but passed through unchanged because all processors were skipped. For example, this occurs when `UnsupportedTemporality` or `DestructiveOnVended` restrictions apply. You can create an alarm on this metric to detect metrics in your selection criteria that cannot be processed.	PipelineName, dataset	Count

Dimensions

CloudWatch pipelines metrics use the following dimensions:

PipelineName

Name of the pipeline.

ErrorType

Type of error encountered.

WarningType

Type of warning encountered.

DataSource (logs pipelines)

Source of the data (AWS service name or third-party source).

DataType (logs pipelines)

Type of data being processed.

ErrorSource (logs pipelines)

Origin of the error (s3, aws.secrets, cloudwatch_logs).

ErrorComponent (logs pipelines)

Component where error occurred (source, sink, extension).

dataset (metrics pipelines)

The OTel metric dataset identifier. Currently always `default`.

Error types

The following error types are tracked in `PipelineErrorsByErrorType`:

`ALL`

The total count of all errors on the pipeline.

`PARSE_FAILURE`

Data parsing errors.

`PROCESSOR_ERRORS`

Processing operation failures.

`ACCESS_DENIED` (logs pipelines)

Permission-related failures.

`RESOURCE_NOT_FOUND` (logs pipelines)

Specified resource doesn't exist.

`SOURCE_READ_FAILURE` (logs pipelines)

Failures reading from source.

`PAYLOAD_SIZE_EXCEEDED` (logs pipelines)

Data size limit exceeded.

Warning types

The following warning types can occur on a pipeline:

`THROTTLED`

Indicates that the volume of data being sent has exceeded existing rate limits, causing some data points or events to be dropped or delayed to protect the system and make sure stability.

`DestructiveOnVended` (metrics pipelines)

A destructive processor attempted to modify a vended metric (`instrumentation_scope.name` starting with `cloudwatch.aws/`). The pipeline skipped the operation and counted the affected datapoints in `PipelineDataPointsUnprocessed`.

`UnsupportedTemporality` (metrics pipelines)

A destructive processor attempted to modify a metric with cumulative aggregation temporality. The pipeline skipped the operation and passed the metric through unchanged. Cumulative temporality means the metric reports values accumulated since a fixed start time (as defined by the OpenTelemetry metrics data model). The pipeline counts the affected datapoints in `PipelineDataPointsUnprocessed`. Monitor this warning to identify metrics in your selection criteria that cannot be processed by destructive operations.

`UnknownOttlPath` (metrics pipelines)

Processor configuration references an unrecognized OTTL path — attribute operation skipped.

Viewing metrics

You can view CloudWatch pipelines metrics using the following methods:

Using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/
2. In the navigation pane, choose **Metrics**
3. Choose the **AWS/Observability Admin** namespace
4. Select the metric dimension to view

Using the AWS CLI

```
aws cloudwatch get-metric-statistics \
  --namespace "AWS/Observability Admin" \
  --metric-name "PipelineBytesIn" \
  --dimensions Name=PipelineName,Value=my-pipeline \
  --start-time "2025-10-29T00:00:00" \
  --end-time "2025-10-29T23:59:59" \
  --period 300 \
  --statistics Sum
```

Creating alarms

You can create CloudWatch alarms based on any of these metrics. Here's an example of creating an alarm for pipeline errors:

```
aws cloudwatch put-metric-alarm \
  --alarm-name "HighPipelineErrors" \
  --alarm-description "Alert on high error rate" \
  --metric-name "PipelineErrors" \
  --namespace "AWS/Observability Admin" \
  --dimensions Name=PipelineName,Value=my-pipeline \
  --period 300 \
  --evaluation-periods 2 \
  --threshold 10 \
  --comparison-operator GreaterThanThreshold \
  --statistic Sum \
  --alarm-actions arn:aws:sns:region:account-id:topic-name
```

Best practices for CloudWatch pipelines metrics

Monitor data flow

- Use `PipelineBytesIn` and `PipelineBytesOut` to track data volume
- Monitor `PipelineRecordsIn` and `PipelineRecordsOut` to track record counts
- Watch for unexpected changes in throughput patterns
- To monitor dropped records, compute `PipelineRecordsIn` - `PipelineRecordsOut` over a given period; a sustained positive difference indicates records that did not reach the destination.

Track errors and warnings

- Create alarms for `PipelineErrors` to detect issues quickly
- Use `PipelineErrorsByErrorType` to diagnose specific problems
- Monitor `PipelineWarnings` to identify potential issues early

Configure appropriate thresholds

- Base thresholds on your expected data patterns
- Account for normal variations in data volume
- Consider peak usage periods when setting alarm thresholds

Troubleshooting

This section covers common issues you might encounter when working with CloudWatch pipelines and their solutions.

Pipeline creation failures

- **Issue:** Pipeline creation fails with permission errors

  Solution: Verify that your IAM role has the required PassRole permissions for any roles specified in the pipeline configuration
- **Issue:** Configuration validation errors

  Solution: Use the ValidateTelemetryPipeline API to check your configuration syntax before creating the pipeline

- **Issue:** Log source not available to select

  Possible reason: There is already a log source with the same data source name and type. Duplicate combinations are greyed out since you can only create one of each.

Data processing issues

- **Issue:** No data flowing through pipeline

  Solution: Check that your data source is properly configured and generating data, and verify that CloudWatch Logs resource policies are correctly set
- **Issue:** Processor errors in pipeline

  Solution: Review processor configuration and make sure input data format matches processor expectations

Amazon CloudWatch managed Prometheus collectors

Amazon CloudWatch managed Prometheus collectors provides a fully managed, agentless scraper, or collector, that automatically discovers and pulls Prometheus-compatible metrics. With Amazon CloudWatch managed Prometheus collectors, you eliminate the operational overhead of running and scaling your own metric collection infrastructure. The collectors automatically discover targets in your VPC, scrape Prometheus-compatible metrics, and deliver them to CloudWatch. You don't have to manage, install, patch, or maintain agents, collectors or scrapers for Prometheus metrics. Managed collectors automatically scale to provide highly available, reliable metric collection for your AWS infrastructure.

A managed collector creates an Elastic Network Interface (ENI) for each subnet that you specify when you create the scraper. The collector scrapes metrics through these ENIs using OpenTelemetry Protocol (OTLP). It delivers the metrics to your CloudWatch dataset through a VPC endpoint. The scraped data never travels on the public internet. You can query metrics in CloudWatch with PromQL (Prometheus Query Language) for flexible and scalable analytics.

> **Note**
>
> Scraping metrics might incur VPC data transfer charges between the collector and your targets. To reduce this network transfer cost, configure your Prometheus `/metrics` endpoint to compress responses (for example, with gzip). Compression lowers the volume

of data that the collector transfers across the network. It does not change the number of metrics that CloudWatch ingests.

We charge for Prometheus collectors by the hour, and CloudWatch OpenTelemetry metric ingestion pricing applies. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Topics

- Integrate Amazon MSK
- Integrate Amazon OpenSearch Service
- Integrate Amazon EKS
- VPC-connected managed collector
- Scraper configuration
- Monitor collectors

Integrate Amazon MSK

With an Amazon CloudWatch managed Prometheus collector, you can automatically discover and collect Prometheus metrics from your Amazon MSK clusters. When you enable Open Monitoring on your Amazon MSK cluster, it exposes Prometheus metrics through the Java Management Extensions (JMX) Exporter and Node Exporter. The managed collector connects to your VPC and scrapes these metrics from all brokers using DNS-based service discovery and delivers these metrics directly to CloudWatch. With this integration, you can monitor host-level metrics, JVM-level metrics, and Kafka broker metrics in CloudWatch without deploying any agents.

Note

When a Amazon CloudWatch managed Prometheus collector delivers your Amazon MSK metrics to CloudWatch, it automatically enriches each metric with attributes that identify its origin. Every metric carries the collector's instrumentation scope, cloud attributes that record the AWS account and Region, and a unit that the collector infers from the metric name. You can filter and group on these attributes when you query your metrics with PromQL.

For instructions on collecting consumer-group lag, committed offsets, and membership, see Collect Amazon MSK consumer-group metrics in CloudWatch. That procedure uses a managed Kafka exporter and IAM authentication. The following procedure collects Open Monitoring metrics.

Prerequisites

This procedure assumes familiarity with Amazon MSK cluster administration and Amazon VPC networking concepts.

- Amazon MSK cluster in provisioned mode, with standard or express brokers. Managed collectors do not support Amazon MSK Serverless.
- Open Monitoring enabled on the cluster. For more information, see Open Monitoring with Prometheus in the *Amazon MSK Developer Guide*.
- At least two subnets in different Availability Zones
- Security group allowing the collector to reach broker ports 11001 and 11002

Step 1: Enable Open Monitoring

Enable Open Monitoring on your Amazon MSK cluster to expose Prometheus metrics. For instructions on enabling Open Monitoring, see Open Monitoring with Prometheus in the *Amazon MSK Developer Guide*.

```
aws kafka update-monitoring \
  --cluster-arn "arn:aws:kafka:us-west-2:123456789012:cluster/my-cluster/abc123-def456"
 \
  --current-version "K1A2B3C4D5" \
  --open-monitoring '{
    "Prometheus": {
      "JmxExporter": {"EnabledInBroker": true},
      "NodeExporter": {"EnabledInBroker": true}
    }
  }'
```

Note

The `--open-monitoring` parameter alone exposes the Prometheus endpoints on ports 11001 and 11002. Enhanced monitoring tiers such as `PER_TOPIC_PER_PARTITION` are

separate from Open Monitoring and can incur additional charges, so set them only if you need that level of Amazon MSK metrics.

Step 2: Get your cluster's DNS name

Amazon MSK provides a cluster-level DNS name that resolves to all broker IPs. Using this for service discovery makes your monitoring resilient to broker replacements and cluster scaling.

Get your cluster DNS name (remove the broker-specific prefix like `b-1.` or `b-2.`):

```
aws kafka get-bootstrap-brokers --cluster-arn "arn:aws:kafka:us-
west-2:123456789012:cluster/my-cluster/abc123-def456"
```

For example, if the bootstrap broker returns `b-1.my-cluster.abc123.c4.kafka.us-west-2.amazonaws.com`, use `my-cluster.abc123.c4.kafka.us-west-2.amazonaws.com` as your cluster DNS name.

Step 3: Configure scrape config

The following is an example scrape configuration for Amazon MSK. You reference this configuration when you create the scraper in the next step. For more information about configuration options, see Scraper configuration.

```
global:
  scrape_interval: 60s
  external_labels:
    cluster_name: my-msk-cluster

scrape_configs:
  - job_name: 'msk-jmx'
    dns_sd_configs:
      - names:
          - my-cluster.abc123.c4.kafka.us-west-2.amazonaws.com
        type: A
        port: 11001
    relabel_configs:
      - source_labels: [__meta_dns_name]
        target_label: broker_dns
      - source_labels: [__address__]
        target_label: instance
```

```
      - target_label: compute_platform
        replacement: 'msk'

  - job_name: 'msk-node'
    dns_sd_configs:
      - names:
          - my-cluster.abc123.c4.kafka.us-west-2.amazonaws.com
        type: A
        port: 11002
    relabel_configs:
      - source_labels: [__meta_dns_name]
        target_label: broker_dns
      - source_labels: [__address__]
        target_label: instance
      - target_label: compute_platform
        replacement: 'msk'
```

Step 4: Create the scraper

Create the scraper with a CloudWatch destination, using the scrape configuration for your Amazon MSK brokers from the previous step.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "msk-metrics-scraper",
  "source": {
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  },
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  },
  "scrapeConfiguration": {
```

```
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "msk-metrics-scraper" \
  --source '{
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  }' \
  --scrape-configuration configurationBlob=$(cat msk-config.yaml | base64 -w 0) \
  --destination '{
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  }'
```

Available metrics

- **JMX Exporter (port 11001)** — Kafka broker internals including topic metrics, partition metrics, request rates, consumer lag, and under-replicated partitions.
- **Node Exporter (port 11002)** — Host-level metrics including CPU, memory, disk I/O, and network throughput.

For a complete list of available metrics, see MSK metrics collected by managed collectors in the *Amazon Managed Service for Prometheus User Guide*.

Validate metrics collection

To confirm that the collector is delivering both JMX and Node Exporter metrics, run the following queries in CloudWatch with Query Studio. If each query returns data points, the collector is scraping the corresponding exporter successfully.

The following query returns a broker-level Kafka metric from the JMX Exporter (port 11001). It reports the mean rate of broker topic activity, such as messages and bytes flowing through the brokers. Data points confirm that the collector is scraping Kafka broker metrics.

```
kafka_server_BrokerTopicMetrics_MeanRate
```

The following query returns host-level CPU utilization from the Node Exporter (port 11002). It subtracts the average idle CPU rate over a five-minute window from 100 to give the percentage of CPU that the brokers are using. Data points confirm that the collector is scraping host-level metrics.

```
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

View the automatic dashboard

After the collector begins delivering your Amazon MSK metrics to CloudWatch, the CloudWatch console provides an automatic dashboard named **MSK OTel**. To open it, see MSK OTel, or sign in to the AWS Management Console, open the CloudWatch console, choose **Dashboards** in the navigation pane, choose **Automatic dashboards**, and then choose **MSK OTel**. You can start monitoring your clusters without building any widgets or dashboards yourself.

If the automatic dashboard meets your needs, you can use it as is. To build a tailored monitoring experience, you can add any of its widgets to a custom dashboard. For more information about custom dashboards, see Using CloudWatch dashboards.

Cross-account observability

For cross-account Amazon MSK monitoring, we recommend using Amazon CloudWatch metric centralization. For more information, see CloudWatch metrics centralization.

For alternative cross-account scraper configurations using role chaining, see Cross-account Amazon MSK integration in the *Amazon Managed Service for Prometheus User Guide*.

Current limitations

- Managed collectors do not support Amazon MSK Serverless clusters.
- Managed collectors do not support public access combined with KRaft metadata mode.
- Each Amazon MSK cluster and CloudWatch dataset combination requires one collector.

Collect Amazon MSK consumer-group metrics in CloudWatch

With an Amazon CloudWatch managed Prometheus collector and a managed Kafka exporter, you can collect consumer-group metrics from your Amazon MSK Provisioned clusters, such as consumer lag, committed offsets, and the number of members in each consumer group. The collector runs the exporter for you and delivers the metrics directly to CloudWatch, so you do not deploy or operate an exporter yourself. The collector connects to your cluster with AWS Identity and Access Management (IAM) authentication, and you grant it access with an Amazon MSK cluster policy.

Consumer-group metrics do not require Open Monitoring. If you also collect JMX or Node Exporter metrics, keep the jobs from Integrate Amazon MSK.

Prerequisites

Before you create a scraper, make sure that you have the following:

- An active Amazon MSK Provisioned cluster with Standard brokers and Apache Kafka version 3.6.0 or later. The cluster must have IAM authentication turned on and be in the same account and AWS Region as the scraper.
- At least two subnets in different Availability Zones in the cluster's VPC, with DNS enabled.
- An inbound rule on the cluster's security group that allows TCP traffic on port 9098 from the security group that you provide for the scraper.
- Permission to create a scraper and to read and change the cluster policy (`kafka:GetClusterPolicy` and `kafka:PutClusterPolicy`).

For additional caller-permission and AWS CLI requirements, see Prerequisites in the Amazon Managed Service for Prometheus User Guide.

This exporter does not support Amazon MSK Serverless. Use the metrics that Amazon MSK sends to CloudWatch for Serverless.

Step 1: Create a scraper with a CloudWatch destination

Save the following YAML scrape configuration as `msk-consumer.yaml`. Replace the cluster ARN. For an existing scraper, add this job to its full configuration and keep any existing jobs. Then update the scraper as described in Update or remove the exporter.

```
global:
  scrape_interval: 30s
scrape_configs:
```

```
  - job_name: kafka-exporter
    relabel_configs:
      - target_label: clusterArn
        replacement: 'arn:aws:kafka:us-west-2:111122223333:cluster/my-cluster/cluster-
id'
    metric_relabel_configs:
      - source_labels: [__name__]
        regex: kafka_consumergroup_(current_offset|members|lag)
        action: keep
```

Keep the job name `kafka-exporter`. The collector configures its scrape target automatically. The `clusterArn` label keeps metrics from different clusters apart when you query them. Its value must match the cluster ARN in your request. The `metric_relabel_configs` rule keeps three consumer-group metrics: lag, committed offsets, and members. The exporter can send more metrics, and Amazon MSK already provides some of that information. For details, see Create a scrape configuration.

Run the following AWS CLI command to create the scraper. Replace the Region, account ID, cluster ARN, subnets, security group, and both filters. The example selects topics that start with `orders` and consumer groups that start with `payments`. To select all topics or all consumer groups, omit the filter. Consumer-group metrics can also include other topics that a selected consumer group consumes. To control which metrics reach CloudWatch, use `metric_relabel_configs` in your scrape configuration.

```
aws amp create-scraper \
  --region us-west-2 \
  --alias msk-consumer-metrics \
  --source '{"vpcConfiguration":{"subnetIds":["subnet-subnet-id-1","subnet-subnet-
id-2"],"securityGroupIds":["sg-security-group-id"]}}' \
  --exporters '[{"kafkaConfiguration":{"clusterArn":"arn:aws:kafka:us-
west-2:111122223333:cluster/my-cluster/cluster-id","topicFilter":"^orders.*
$","consumerGroupFilter":"^payments.*$"}}]' \
  --cli-binary-format base64 \
  --scrape-configuration "configurationBlob=$(base64 < msk-consumer.yaml | tr -d '\n')"
 \
  --destination '{"cloudWatchConfiguration":{"datasetArn":"arn:aws:cloudwatch:us-
west-2:111122223333:dataset/default"}}'
```

Record the scraper ID. Check its status with the `aws amp describe-scraper` AWS CLI command, and continue when `scraper.status.statusCode` is `ACTIVE`. The scraper collects no Kafka metrics until you grant it access to the cluster in the next step.

Step 2: Grant the scraper access to the cluster

Add an Amazon MSK cluster policy that allows the scraper's service-linked role to connect to the cluster and to describe its topics and consumer groups. After you add the policy, collection starts on its own, and the first metrics can take a few minutes to appear. You do not need to update the scraper.

Grant the scraper's service-linked role. Do not use an account root principal or a role ARN with a wildcard. Amazon MSK accepts a root principal, but it grants the scraper nothing, and it rejects a wildcard role ARN.

To add the cluster policy

1. Get the scraper's role ARN:

```
aws amp describe-scraper \
  --region us-west-2 \
  --scraper-id scraper-id \
  --query scraper.roleArn --output text
```

2. Save the following policy as `policy.json`. Replace the role ARN, Region, account ID, cluster name, and cluster ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "AWS": "scraper-role-arn" },
      "Action": "kafka-cluster:Connect",
      "Resource": "arn:aws:kafka:us-west-2:111122223333:cluster/my-cluster/cluster-
id"
    },
    {
      "Effect": "Allow",
      "Principal": { "AWS": "scraper-role-arn" },
      "Action": "kafka-cluster:DescribeTopic",
      "Resource": "arn:aws:kafka:us-west-2:111122223333:topic/my-cluster/cluster-
id/*"
    },
    {
      "Effect": "Allow",
```

```
      "Principal": { "AWS": "scraper-role-arn" },
      "Action": "kafka-cluster:DescribeGroup",
      "Resource": "arn:aws:kafka:us-west-2:111122223333:group/my-cluster/cluster-
id/*"
    }
  ]
}
```

3. Check whether the cluster already has a policy:

```
aws kafka get-cluster-policy \
  --region us-west-2 \
  --cluster-arn arn:aws:kafka:us-west-2:111122223333:cluster/my-cluster/cluster-id
```

   A `NotFoundException` error means that the cluster has no policy.

4. If the cluster has no policy, add yours:

```
aws kafka put-cluster-policy \
  --region us-west-2 \
  --cluster-arn arn:aws:kafka:us-west-2:111122223333:cluster/my-cluster/cluster-id
 \
  --policy file://policy.json
```

   If the cluster already has a policy, `put-cluster-policy` replaces it. The `Policy` value from `get-cluster-policy` is the existing policy as a JSON string. Add its statements to the `Statement` list in `policy.json`. Then run `put-cluster-policy` with `--current-version` and the `CurrentVersion` value from `get-cluster-policy`.

To grant every scraper in the account, to limit the topics and groups, or to see the full merge command, see Grant the scraper access to the cluster in the Amazon Managed Service for Prometheus User Guide.

Step 3: Validate metrics collection

To confirm that the collector is delivering consumer-group metrics, choose the scraper's Region and run each of the following queries separately in CloudWatch with Query Studio. Replace the cluster ARN.

```
kafka_consumergroup_members{"@instrumentation.cloudwatch.managed-collector.scraper-
type"="msk", clusterArn="arn:aws:kafka:us-west-2:111122223333:cluster/my-
cluster/cluster-id"}
```

```
kafka_consumergroup_current_offset{"@instrumentation.cloudwatch.managed-
collector.scraper-type"="msk", clusterArn="arn:aws:kafka:us-
west-2:111122223333:cluster/my-cluster/cluster-id"}
```

```
kafka_consumergroup_lag{"@instrumentation.cloudwatch.managed-collector.scraper-
type"="msk", clusterArn="arn:aws:kafka:us-west-2:111122223333:cluster/my-
cluster/cluster-id"}
```

A matching group must exist for its metrics to appear. The first metrics can take a few minutes to appear after you add the cluster policy.

If the scraper is `ACTIVE` but no `kafka_consumergroup_*` metrics appear under the service name `kafka-exporter`, check the cluster policy. Confirm that it grants the three actions to the scraper's current service-linked role ARN. If you re-created the scraper, its role ARN changed. If you use the account-wide policy, confirm that every statement keeps both `StringEquals` on `aws:PrincipalAccount` and `ArnLike` on `aws:PrincipalArn`. You do not need to update the scraper after you fix the policy.

For other checks, see Validate metrics collection. Those checks cover the cluster policy, caller permissions, IAM broker connectivity, collection filters, and metric labels. The up query on that page applies only to an Amazon Managed Service for Prometheus workspace destination. If the cluster policy is correct and metrics still do not appear on a cluster with many topics and consumer groups, try narrowing `topicFilter` and `consumerGroupFilter`.

For information about these metrics, including the `-1` value when a group has no committed offset for a topic and partition, see Consumer-group metrics.

Update or remove the exporter

Omitting `exporters` in an `update-scraper` request keeps the current exporter. An empty list removes it. Removing the cluster policy does not stop collection right away. First remove the exporter or delete the scraper, and confirm with `aws amp describe-scraper` that the scraper status is `ACTIVE` and the exporter is gone, or that the scraper no longer exists. Then remove the policy statements. For the commands, see Update or remove the exporter.

Integrate Amazon OpenSearch Service

With an Amazon CloudWatch managed Prometheus collector, you can automatically collect Prometheus-compatible metrics from your Amazon OpenSearch Service domains. The managed collector connects to your VPC, collects cluster, node, and index metrics from the domain, and delivers these metrics directly to CloudWatch. With this integration, you can monitor the health and performance of your domains in CloudWatch without deploying any agents or exporters.

> **Note**
>
> When a Amazon CloudWatch managed Prometheus collector delivers your Amazon OpenSearch Service metrics to CloudWatch, it automatically enriches each metric with attributes that identify its origin. Every metric carries the collector's instrumentation scope, cloud attributes that record the AWS account and Region, and a unit that the collector infers from the metric name. You can filter and group on these attributes when you query your metrics with PromQL.

Prerequisites

This procedure assumes that you are familiar with Amazon OpenSearch Service domain administration and Amazon VPC networking concepts.

- An Amazon OpenSearch Service domain with VPC access. Managed collectors support only domains that have VPC access. Domains with public access are not supported.
- At least two subnets in different Availability Zones
- Security group rules that allow the collector to reach your domain endpoint over HTTPS (port 443). Add an inbound rule to the domain's security group that allows HTTPS traffic from the security group that you provide for the collector.

Step 1: Configure scrape configuration

The following is an example scrape configuration for Amazon OpenSearch Service. The managed collector connects to the domain that you specify and collects its metrics automatically, so you do not specify scrape targets in the configuration. The configuration must include a `scrape_configs` section with a job whose `job_name` is exactly `opensearch-exporter`. You

reference this configuration when you create the scraper in the next step. For more information about configuration options, see Scraper configuration.

```
global:
  external_labels:
    domain_name: my-opensearch-domain

scrape_configs:
  - job_name: opensearch-exporter
    scrape_interval: 60s
```

Step 2: Create the scraper

Create the scraper with a CloudWatch destination. You specify the domain to collect from in the `exporters` field, and provide the networking (subnets and security group) in the `source` field.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the subnet, security group, domain, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "opensearch-metrics-scraper",
  "source": {
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  },
  "exporters": [
    {
      "openSearchConfiguration": {
        "domainArn": "arn:aws:es:us-west-2:123456789012:domain/my-opensearch-domain"
      }
    }
  ],
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  },
```

```
  "scrapeConfiguration": {
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the subnet, security group, domain, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "opensearch-metrics-scraper" \
  --source '{
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  }' \
  --exporters '[
    {
      "openSearchConfiguration": {
        "domainArn": "arn:aws:es:us-west-2:123456789012:domain/my-opensearch-domain"
      }
    }
  ]' \
  --scrape-configuration configurationBlob=$(base64 -w 0 opensearch-config.yaml) \
  --destination '{
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  }'
```

Available metrics

- **Cluster metrics** — Cluster health and status, number of nodes and data nodes, active and relocating shards, and pending tasks.
- **Node metrics** — CPU usage, JVM heap usage and garbage collection, operating system memory, disk usage, and thread pool activity for each node.
- **Index metrics** — Indexing and search rates and latencies, document counts, merge and refresh activity, and cache usage.

For a complete list of available metrics, see Metrics collected from Amazon OpenSearch Service in the *Amazon Managed Service for Prometheus User Guide*.

Validate metrics collection

To confirm that the collector is delivering metrics from your domain, run the following query in CloudWatch with Query Studio. The query returns the cluster health status for your domain. If it returns data points, the collector is collecting metrics from your domain successfully.

```
opensearch_cluster_health_status
```

Cross-account observability

For cross-account Amazon OpenSearch Service monitoring, we recommend using Amazon CloudWatch metric centralization. For more information, see CloudWatch metrics centralization.

For more information about alternative cross-account scraper configurations using role chaining, see Integrate Amazon OpenSearch Service in the *Amazon Managed Service for Prometheus User Guide*.

Current limitations

- Managed collectors support only domains with VPC access.
- A scraper collects metrics from a single Amazon OpenSearch Service domain. To collect metrics from more than one domain, create a separate scraper for each domain.

Integrate Amazon EKS

An Amazon CloudWatch managed Prometheus collector scrapes metrics that are Prometheus-compatible. For more information about Prometheus-compatible metrics, see What are Prometheus-compatible metrics?. Amazon EKS clusters expose metrics for the API server. Amazon EKS clusters that are Kubernetes version 1.28 or above also expose metrics for the kube-scheduler and kube-controller-manager. For more information, see Fetch control plane raw metrics in Prometheus format in the *Amazon EKS User Guide*.

The *Amazon Managed Service for Prometheus User Guide* provides detailed setup instructions. For instructions on configuring Amazon EKS clusters, access entries, and ConfigMap setup, see Set up managed collectors for Amazon EKS.

Note

The collector enriches each Amazon EKS metric with attributes that identify its origin — the instrumentation scope, the AWS account and Region, an inferred unit, and the cluster name and cluster ARN. You can filter and group on these attributes when you query your metrics with PromQL.

Create a scraper with CloudWatch destination

You can use GetDefaultScraperConfiguration to retrieve a general-purpose scraper configuration, or provide your own.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the AWS account, security group, subnet, Amazon EKS cluster, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "eks-metrics-scraper",
  "source": {
    "eksConfiguration": {
      "clusterArn": "arn:aws:eks:us-west-2:123456789012:cluster/my-cluster",
      "securityGroupIds": ["sg-security-group-id"],
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"]
    }
  },
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  },
  "scrapeConfiguration": {
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the AWS account, security group, subnet, Amazon EKS cluster, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "eks-metrics-scraper" \
  --source eksConfiguration="{clusterArn='arn:aws:eks:us-
west-2:123456789012:cluster/my-cluster', \
    securityGroupIds=['sg-security-group-id'], \
    subnetIds=['subnet-subnet-id-1','subnet-subnet-id-2']}" \
  --scrape-configuration configurationBlob=$(cat eks-config.yaml | base64 -w 0) \
  --destination cloudWatchConfiguration="{datasetArn='arn:aws:cloudwatch:us-
west-2:123456789012:dataset/default'}"
```

View the automatic dashboard

After the collector begins delivering your Amazon EKS metrics to CloudWatch, the CloudWatch console provides an automatic dashboard named **EKS OTel**. To open it, see EKS OTel, or sign in to the AWS Management Console, open the CloudWatch console, choose **Dashboards** in the navigation pane, choose **Automatic dashboards**, and then choose **EKS OTel**. You can start monitoring your clusters without building any widgets or dashboards yourself.

If the automatic dashboard meets your needs, you can use it as is. To build a tailored monitoring experience, you can add any of its widgets to a custom dashboard. For more information about custom dashboards, see Using CloudWatch dashboards.

Cross-account observability

For cross-account Amazon EKS monitoring, we recommend using Amazon CloudWatch metric centralization. For more information, see CloudWatch metrics centralization.

For alternative cross-account scraper configurations, see Cross-account scrapers in the *Amazon Managed Service for Prometheus User Guide*.

VPC-connected managed collector

The VPC-connected Amazon CloudWatch managed Prometheus collector scrapes Prometheus-compatible metrics from any resource reachable within your VPC — Amazon EC2 instances and

Amazon ECS tasks. You provide subnets and a security group; the collector creates elastic network interfaces (ENIs) and scrapes Prometheus `/metrics` endpoints according to your configuration.

For detailed, workload-specific walkthroughs, see the CloudWatch observability solutions for Prometheus metric collection: Amazon CloudWatch solution: Prometheus metric collection on Amazon Amazon EC2 and Amazon CloudWatch solution: Prometheus metric collection on Amazon ECS.

Prerequisites

This procedure assumes familiarity with Amazon VPC subnets, security groups, and Prometheus exporter endpoints.

- Amazon VPC with DNS enabled
- At least two subnets in different Availability Zones
- Security group allowing the collector to reach your target exporter ports
- Targets that expose a Prometheus-compatible `/metrics` endpoint

The collector delivers scraped metrics to CloudWatch over the AWS network, without traversing the public internet. The subnets and security groups that you specify provide connectivity to your scrape targets; they do not provide the CloudWatch delivery path. You do not need to configure internet access, a NAT gateway, or a CloudWatch interface VPC endpoint in your VPC for metric delivery.

Create a scraper

You can use GetDefaultScraperConfiguration to retrieve a general-purpose scraper configuration, or provide your own.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "vpc-metrics-scraper",
  "source": {
```

```
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  },
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  },
  "scrapeConfiguration": {
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "vpc-metrics-scraper" \
  --source '{
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  }' \
  --scrape-configuration configurationBlob=$(cat scrape-config.yaml | base64 -w 0) \
  --destination '{
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  }'
```

Collecting metrics from Amazon EC2

To scrape metrics from Amazon EC2 instances running Prometheus exporters such as DCGM Exporter or Node Exporter, use `static_configs` with the instance private IP addresses:

```
global:
```

```
  scrape_interval: 60s

scrape_configs:
  - job_name: 'ec2-node-exporter'
    static_configs:
      - targets:
          - '10.0.1.10:9100'
          - '10.0.1.11:9100'
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
      - target_label: compute_platform
        replacement: 'ec2'

  - job_name: 'ec2-dcgm-exporter'
    static_configs:
      - targets:
          - '10.0.1.10:9400'
          - '10.0.1.11:9400'
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
      - target_label: compute_platform
        replacement: 'ec2'
```

Collecting metrics from Amazon ECS

For Amazon ECS tasks that you register with AWS Cloud Map, use DNS service discovery to automatically find and scrape your containers:

```
global:
  scrape_interval: 60s

scrape_configs:
  - job_name: 'ecs-services'
    dns_sd_configs:
      - names:
          - 'my-service.my-namespace.local'
        type: A
        port: 9090
    relabel_configs:
      - source_labels: [__meta_dns_name]
        target_label: service_name
```

```
      - source_labels: [__address__]
        target_label: instance
      - target_label: compute_platform
        replacement: 'ecs'
```

Cross-account observability

For cross-account VPC monitoring, we recommend using Amazon CloudWatch metric centralization. For more information, see CloudWatch metrics centralization.

For alternative cross-account scraper configurations, see Cross-account scrapers in the *Amazon Managed Service for Prometheus User Guide*.

Security best practices

- Specify private subnets without direct internet access for the collector ENIs.
- Restrict security group ingress to the scraper security group on specific exporter ports only.
- Enable TLS encryption in transit for all exporter endpoints where possible.

Scraper configuration

Amazon CloudWatch managed Prometheus collectors use a Prometheus-compatible YAML configuration to define how the collector scrapes metrics. You provide this configuration as a base64-encoded blob when creating or updating a scraper. The configuration uses standard Prometheus scrape configuration format, with some limitations specific to managed collectors.

You can use GetDefaultScraperConfiguration to retrieve a general-purpose scraper configuration, or provide your own.

Supported configuration sections

The following top-level sections of the Prometheus scrape configuration are supported:

- `global` — Global settings including `scrape_interval`, `scrape_timeout`, and `external_labels`.
- `scrape_configs` — A list of scrape jobs, each defining targets and how to scrape them.

Within each `scrape_configs` entry, the following fields are supported:

- `job_name` — A unique name for the scrape job.
- `metrics_path` — The HTTP path to scrape (defaults to `/metrics`).
- `scheme` — The protocol scheme (`http` or `https`).
- `scrape_interval` — Per-job override of the global scrape interval.
- `scrape_timeout` — Per-job override of the global scrape timeout.
- `static_configs` — Static list of targets.
- `dns_sd_configs` — DNS-based service discovery.
- `relabel_configs` — Relabeling rules applied before scraping.
- `metric_relabel_configs` — Relabeling rules applied after scraping.
- `tls_config` — TLS configuration for HTTPS targets.

Example configuration

The following example demonstrates a basic scrape configuration:

```
global:
  scrape_interval: 60s
  scrape_timeout: 10s
  external_labels:
    environment: production

scrape_configs:
  - job_name: 'my-service'
    metrics_path: /metrics
    scheme: http
    static_configs:
      - targets:
          - '10.0.1.10:9090'
          - '10.0.1.11:9090'
        labels:
          team: platform
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
    metric_relabel_configs:
      - source_labels: [__name__]
        regex: 'go_.*'
        action: drop
```

Limitations

- The minimum `scrape_interval` is 30 seconds.
- Managed collectors do not support file-based service discovery (`file_sd_configs`).
- Managed collectors do not support Consul, Eureka, or other external service discovery mechanisms.
- Remote write and remote read configurations do not apply, because the collector handles delivery to CloudWatch.
- The maximum configuration size is 256 KB.

Troubleshooting

If your scraper is not collecting metrics as expected, check the following:

- Verify that the security group attached to the scraper allows outbound traffic to your target ports.
- Confirm that your targets are reachable from the subnets that you specified in the scraper configuration.
- Check that DNS names used in `dns_sd_configs` resolve correctly within the VPC.
- Make sure your targets respond on the configured `metrics_path` with valid Prometheus exposition format.

For additional troubleshooting guidance, see Troubleshooting managed collectors in the *Amazon Managed Service for Prometheus User Guide*.

Monitor collectors

Amazon CloudWatch managed Prometheus collectors automatically vend logs to CloudWatch Logs so you can monitor scraper health, diagnose connectivity issues, and track scraping activity. These logs include information about target discovery, scrape successes and failures, and configuration errors. For more information about collector logging, see Logging for managed collectors in the *Amazon Managed Service for Prometheus User Guide*.

List scrapers

Use the following command to list all scrapers in your account and Region:

```
aws amp list-scrapers
```

To filter scrapers by alias:

```
aws amp list-scrapers --filters alias=my-scraper-alias
```

Describe a scraper

To view the details and current status of a specific scraper:

```
aws amp describe-scraper --scraper-id scraper-id
```

Delete a scraper

To delete a scraper that is no longer needed:

```
aws amp delete-scraper --scraper-id scraper-id
```

> **Note**
>
> Deleting a scraper removes the ENIs that the collector created in your VPC. Allow a few minutes for the deletion to complete and for CloudWatch to clean up all associated resources.

CloudWatch Omni

CloudWatch Omni is an AI-powered observability experience in Amazon CloudWatch for your applications and AI agents, either alone or together. You reach Omni through a dedicated URL for your organization and sign in with the identities you already manage; Omni is accessible outside the AWS Management Console.

Your applications and your agents, together. Omni is built on OpenTelemetry. Telemetry you already send to CloudWatch appears in Omni with nothing to reconfigure, and any other workload you instrument with OpenTelemetry sends its telemetry to an OpenTelemetry Protocol (OTLP) endpoint. Work in the Omni web UI, or in the IDE extension for VS Code, Kiro, and Cursor while you develop AI agents.

Answers before you build a query. Without manual configuration, Omni discovers your services and their dependencies from your telemetry and reports the RED metrics (the request rate, errors, and duration) of each one. Ask a question in plain language and Omni writes the query for you, in SQL or PromQL (Prometheus Query Language). The Omni agent is Omni's built-in AI assistant, and it answers those questions from your own telemetry. When an alert fires, you can start an investigation with the AWS DevOps Agent to find the root cause. Alerts reach your team in Slack, or any destination of your choice through Amazon Simple Notification Service (Amazon SNS).

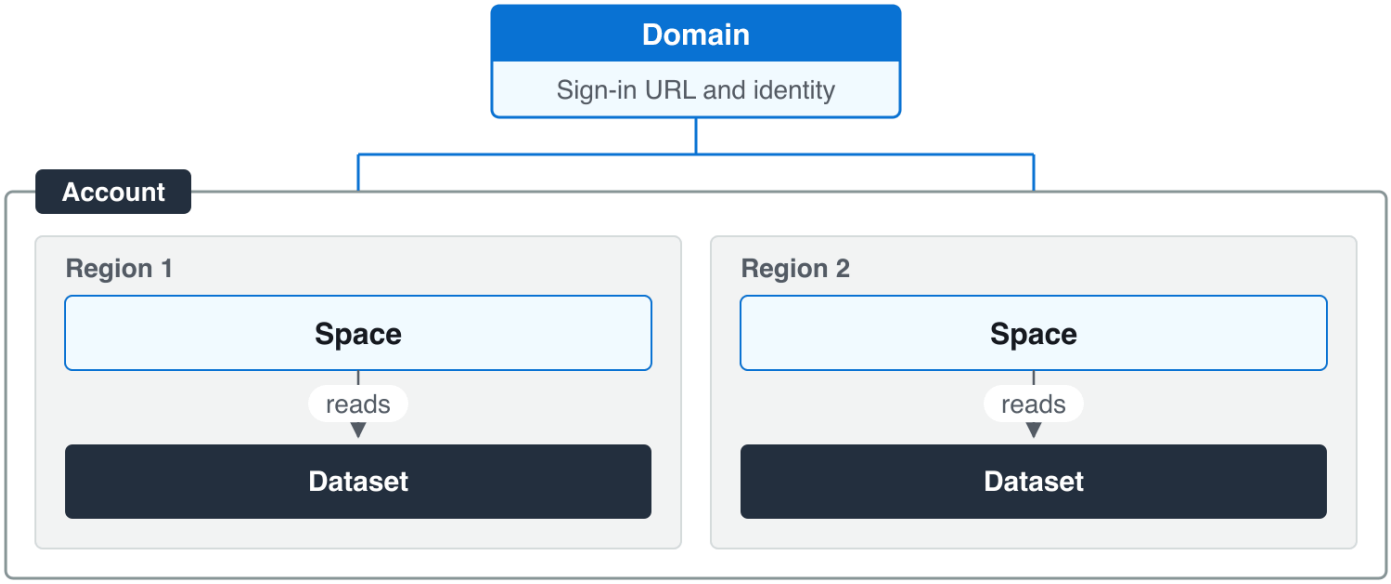
Quality signals for agents, alongside the operational ones. Omni scores your agents' responses with evaluators — continuously in production, and on demand against a dataset. This catches failures that error rates do not show: an agent can return a well-formed response that is wrong. The scores appear alongside latency, errors, and token usage on the same trace. You can bring your own evaluators, including custom criteria scored by a judge model.

How CloudWatch Omni is organized

Three concepts determine where your telemetry lives and who can reach it.

- **Domain** — your organization's entry point. You choose a domain name during setup, and that name becomes your team's sign-in URL, in the format `https://<domain-name>.cloudwatch-omni.global.app.aws`. Record your domain name and its URL — the name is what identifies your domain when you sign in. Your identity provider connects at the domain, not per space. See the section called “Set up Omni”.

- **Space** — where you work. A space gives you access to telemetry from the account and Region in which the space is hosted, and controls who can see it. Because a space maps to one account and Region, that boundary is what separates spaces, rather than a filter applied when you query. You create a space inside your domain, and one domain can contain multiple spaces. To observe multiple accounts or Regions, see the section called "Set up Omni for your organization".
- **CloudWatch Dataset** — how Omni brings your telemetry together. A Dataset makes your logs, metrics, and traces queryable and correlated across signals in one place. You continue to manage ingestion and storage through existing CloudWatch endpoints and resources (for example, log groups). Enabling Omni creates one Dataset for your space.



What you can do in CloudWatch Omni

The following sections describe what you can do in CloudWatch Omni:

- Trace, evaluate, and debug AI agents. See the section called "Monitor AI agents".
- View your services, their dependencies, and their health, discovered automatically from your telemetry. See the section called "Monitor applications with the application map".
- Ask questions about your telemetry in plain language. See the section called "Ask the Omni agent".
- Query across logs, metrics, and traces. See the section called "Explore and query your telemetry".
- Build shared views of your systems. See the section called "Dashboards".

- Investigate with your team and keep a record of what you find. See the section called “Threads”.
- Receive a notification when a condition you define is met. See the section called “Alerts”.
- Investigate incidents to find the root cause. See the section called “Investigate with the AWS DevOps Agent”.
- Connect Omni to Slack. See the section called “Connect Slack”.
- Call Omni from code or a coding agent. See the section called “Use Omni skills”.

How Omni relates to CloudWatch

Omni is an interface and set of workflows built on Amazon CloudWatch. It is not a replacement, and enabling it does not change anything you already run.

- **Built on CloudWatch data.** Omni reads from the same CloudWatch data stores you use today. Your existing console pages, metrics, alarms, dashboards, Logs Insights queries, and APIs keep working unchanged.
- **No re-instrumentation.** Instrumentation you already run (the CloudWatch agent, OTLP pipelines, and AWS SDKs) keeps sending telemetry without changes, and that telemetry appears in Omni.
- **Centralization stays with CloudWatch.** Omni does not aggregate telemetry across accounts and Regions on its own. Telemetry you centralize with CloudWatch centralization rules appears in your space like any other telemetry. See the section called “Set up Omni for your organization”.

You continue to configure how telemetry is collected and stored (log groups, retention, and ingestion endpoints) in the CloudWatch console.

Get started

To try Omni before you set anything up, open the playground, a shared environment with sample data that requires no AWS account or setup. Choose **Explore the playground** on the Omni page in the CloudWatch console, or **Try playground** on the sign-in page.

To set up Omni, you need an AWS account and permission to enable Omni and create a domain and space in the CloudWatch console. For the full prerequisites, see the section called “Set up Omni for a single account”.

To set up Omni, complete the following steps:

1. Set up your domain and space. See the section called “Set up Omni”.
2. Send telemetry from your applications or agents. See the section called “Send telemetry to CloudWatch Omni”.
3. Follow an end-to-end tutorial. See the section called “Tutorial: Fix a production quality regression” for an AI agent, or the section called “Tutorial: Monitor an application” for an application.

Set up Omni

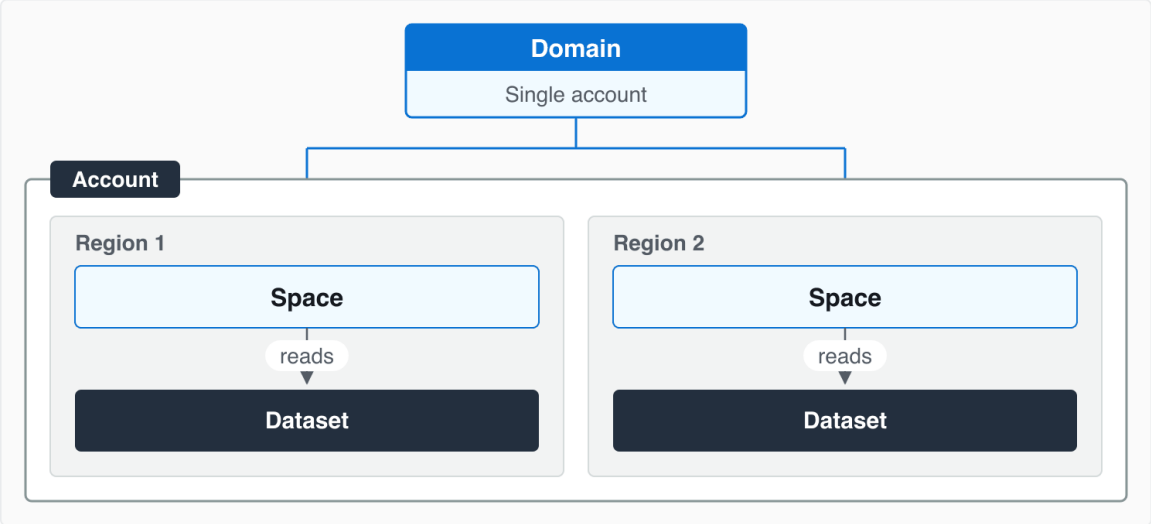
Setting up CloudWatch Omni involves two steps: creating a **domain** (the identity and access boundary for your organization) and creating a **space** (where you work with your telemetry). Depending on your scenario, these steps might happen together in one session or separately across teams.

> **Note**
>
> Enabling Omni does not change or disable anything in CloudWatch. Your existing metrics, logs, alarms, dashboards, and Logs Insights queries keep working.

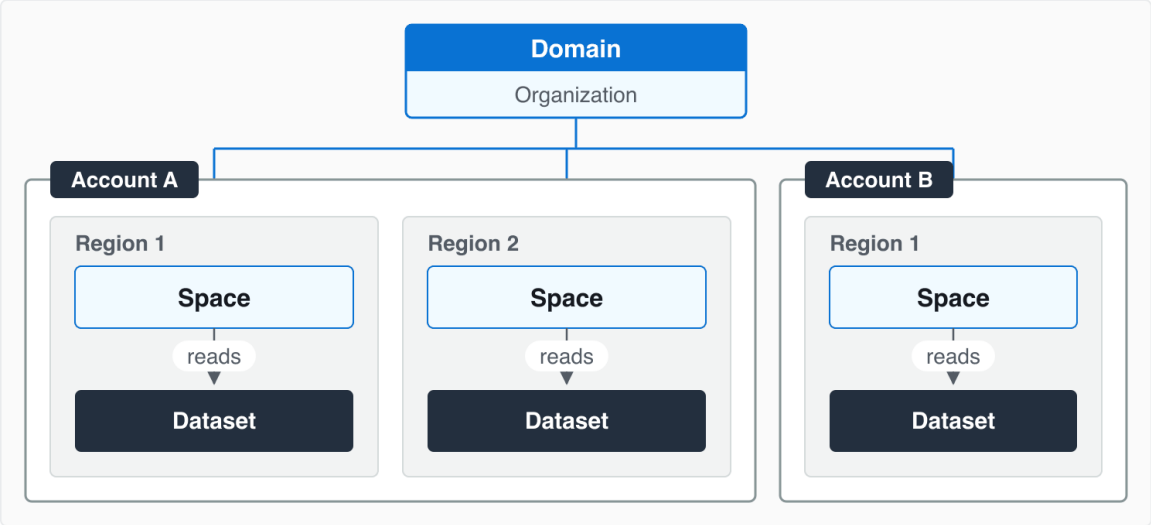
Choose your setup path

How you set up Omni depends on whether you work in a single AWS account or across an AWS Organization.

Account-level domain



Organization-level domain



One space per account and Region

Capability	Single account	Multiple accounts (AWS Organizations)
Single sign-in URL for your organization	Per account	Yes
Member accounts create spaces without additional domain setup	No — each account creates its own domain	Yes

Capability	Single account	Multiple accounts (AWS Organizations)
Centralized identity provider configuration (IAM Identity Center)	Yes — configured per domain	Yes — configured once for the organization
Multiple spaces across accounts share one domain	No	Yes
Accounts that can create this domain	Any individual AWS account	AWS Organizations management account

Choose **single account** setup if you operate in one AWS account. Choose **organization** setup if you use multiple AWS accounts and want a single entry point.

Single account

You create an Account domain and a space together in your AWS account.

For instructions, see the section called “Set up Omni for a single account”.

Multiple accounts (AWS Organizations)

An administrator creates the domain once from the management account. Any member account can then create its own space, which is automatically linked to the domain.

This path requires **trusted access** between AWS Organizations and CloudWatch, enabled for your organization.

For prerequisites and instructions, see the section called “Set up Omni for your organization”.

Your organization already has a domain

If an Organization domain already exists for your AWS Organization, you do not need to create a domain. Your account is already linked. Skip to space setup in the section called “Set up Omni for your organization”.

After setup

After you complete either setup path:

- Send telemetry from your applications or agents. See the section called “Send telemetry to CloudWatch Omni”.
- Add your team and choose what each person can do. See the section called “Control access to your space”.
- Bring telemetry from more than one account into your space. See the centralization section of the section called “Set up Omni for your organization”.

Set up Omni for a single account

Use this guide to set up CloudWatch Omni in a single AWS account. You create an Account domain and a space together in one console session. When you finish, you have a working Omni space ready to receive telemetry.

If your organization uses multiple AWS accounts, see the section called “Set up Omni for your organization” instead.

Prerequisites

- An AWS account with permission to access Amazon CloudWatch in the console.
- Permission to enable Omni and create a domain and space. Grant this to administrators only.
- Sign in to the AWS Region in which you want to enable Omni.
- If you plan to use IAM Identity Center for sign-in, its instance must be in the same AWS Region as your domain.

Choose your domain name

Your domain name becomes part of a URL your whole team uses, and it identifies your organization in Omni.

The name must meet these requirements:

- 3–63 characters using lowercase letters, numbers, and hyphens.
- Must start and end with a letter or number.
- Cannot contain consecutive hyphens.
- Certain names are reserved, including AWS service names, Region-like names, and names beginning with `aws-`, `amazon-`, `cloudwatch-`, or `omni-`.

- Must be unique across all CloudWatch domains.

After you create the domain, it is reachable at `https://<domain-name>.cloudwatch-omni.global.app.aws`.

Set up your domain and space

In the Account domain flow, domain creation and space creation happen together in one session.

1. Open the Omni setup page in the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/home#settings:/omni.
2. Choose **Get started**.
3. Select **Account domain**.
4. Enter your domain name.
5. Complete the permissions step. Omni creates the IAM roles it needs on your behalf.
6. When domain creation finishes, space setup continues automatically in the same flow.
7. Enter a space name and confirm the account and Region.
8. Complete the setup. Omni enables the capabilities described later on this page for your account and Region.

Connect your identity provider (optional)

You configure sign-in for your domain through AWS IAM Identity Center, so your team can use the identities you already manage, including an external provider such as Okta or Microsoft Entra ID connected to IAM Identity Center. You configure this at the domain, not per space.

> **Important**
>
> Your IAM Identity Center instance and your domain must be in the **same AWS Region**. If they are not, you cannot configure sign-in for the domain. Enable IAM Identity Center multi-Region replication to include the Region in which your Omni space is hosted.

If you do not connect an identity provider, your team can sign in to Omni using an IAM user or IAM role (for example, through a deep link from the CloudWatch console).

Capabilities enabled when you create a space

Setting up a space enables the following CloudWatch capabilities in your account and Region. You choose the optional capabilities during setup.

Capability	What it does	Learn more
CloudWatch Dataset integration	Enables forwarding of your logs and traces to the CloudWatch Dataset, which makes them queryable and correlated together in Omni. You do not manage a separate store or change how you manage existing CloudWatch resources such as log groups.	*CloudWatch Omni*
OTel metrics enrichment	Enriches AWS metrics with resource attributes in OpenTelemetry format and makes them queryable with PromQL.	CloudWatch OTel metrics enrichment
OTel span ingestion	Enables trace and span ingestion so Omni can receive and correlate distributed traces.	Enable CloudWatch transaction search

These capabilities apply to telemetry received after you complete space setup. Enabling OTel metrics enrichment and Transaction Search follows existing pricing for those capabilities. See Amazon CloudWatch pricing.

On the first enablement, Omni backfills your space with up to 7 days of existing CloudWatch logs and traces. Telemetry older than 7 days, and data encrypted with a customer managed AWS KMS key, are not included in the backfill.

IAM roles created during setup

Space setup creates the following IAM roles in your account. Omni uses these roles to read and correlate your telemetry. Setup adds a numeric suffix to each role name.

IAM role	Purpose
`CloudWatchOmniSpaceAccessRole`	Assumed by Omni to manage your space, query telemetry, and configure integrations.
`CloudWatchOmniDatasetIntegrationExecutionRole`	Assumed by CloudWatch to make your telemetry available in Omni through the Dataset.
`AgentCoreEvaluationRole`	Assumed by Amazon Bedrock AgentCore to run online AI agent evaluations on your telemetry. This role is optional: during setup you keep the default role, choose an existing role, or opt out. See the section called “Online evaluations”.

For the trust policies of these roles, see the section called “Cross-service confused deputy prevention”.

Configure encryption (optional)

By default, Omni manages the encryption of the data it stores for your space. To control encryption with your own key, configure a customer managed AWS KMS key on the space (when you create the space, or later by updating the space's encryption configuration).

The key must meet these requirements:

- A symmetric encryption key (`ENCRYPT_DECRYPT` key usage), and enabled.
- In the same AWS account and Region as the space.
- The identity attaching the key must have `kms:Decrypt` and `kms:GenerateDataKey` permission on it. Both are checked when you attach the key, so a key granting only one is rejected at attach time rather than failing later on reads or writes. The error names the missing permission.

Data Omni stores for your space, such as dashboards, is encrypted under the key.

The key's own key policy must also allow CloudWatch Omni to use the key. For the statements the service needs and a complete sample key policy, see the section called “Data encryption in CloudWatch Omni”.

Next steps

You now have a working Omni space.

- **Send telemetry** from your applications or agents. See the section called “Send telemetry to CloudWatch Omni”.
- **Add your team** and choose what each person can do. See the section called “Control access to your space”.

Set up Omni for your organization

Use this guide to set up CloudWatch Omni across an AWS Organization. An administrator creates an Organization domain once from the management account, and then any member account can create its own space, automatically linked to the domain.

This provides a single sign-in URL for your organization and centralized identity provider configuration, and removes the need for each account to configure its own domain.

If you operate in a single AWS account, see the section called “Set up Omni for a single account” instead.

> **Note**
>
> Enabling Omni does not change or disable anything in CloudWatch. Your existing metrics, logs, alarms, dashboards, and Logs Insights queries keep working.

Who does what

Organization domain setup involves two roles:

- **Organization administrator** — Creates the domain from the AWS Organizations management account. Chooses the domain name and configures the identity provider.

- **Account owner (member account)** — Creates a space in their own account and Region. The space is automatically linked to the Organization domain. Each member account does this independently.

Part 1: Create the domain (Organization administrator)

Prerequisites

- Access to the AWS Organizations **management account** with permission to enable Omni and create a domain.
- **Trusted access** between AWS Organizations and CloudWatch, enabled for your organization. Enabling trusted access requires the `organizations:EnableAWSServiceAccess` and `iam:CreateServiceLinkedRole` permissions. See Amazon CloudWatch and AWS Organizations.
- The AWS Region you want to work in.
- **Note:** If you plan to use IAM Identity Center, the Organization domain must be created in the same Region as the primary Region of IAM Identity Center. If your member accounts need to create spaces in other Regions, enable IAM Identity Center multi-Region replication to include your desired Regions.

Choose your domain name

Your domain name becomes part of a URL your whole team uses, and it identifies your organization in Omni.

The name must meet these requirements:

- 3–63 characters using lowercase letters, numbers, and hyphens.
- Must start and end with a letter or number.
- Cannot contain consecutive hyphens.
- Certain names are reserved, including AWS service names, Region-like names, and names beginning with `aws-`, `amazon-`, `cloudwatch-`, or `omni-`.
- Must be unique across all CloudWatch domains.

After you create the domain, it is reachable at `https://<domain-name>.cloudwatch-omni.global.app.aws`.

Create the domain

1. Sign in to the **AWS Organizations management account**.
2. Open the Omni setup page in the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/home#settings:/omni.
3. Choose **Get started**.
4. Select **Organization domain**.
5. Enter your domain name.
6. Complete the permissions step. Omni creates the IAM roles it needs on your behalf.
7. When domain creation finishes, record your **domain name** and **domain URL**.

Connect your identity provider

You configure sign-in for your domain through AWS IAM Identity Center, so your team uses the identities you already manage, including an external provider such as Okta or Microsoft Entra ID connected to IAM Identity Center. You configure this at the domain, not per space.

> **Important**
>
> Your IAM Identity Center instance and your domain must be in the **same AWS Region**. If they are not, you cannot configure sign-in for the domain. Enable IAM Identity Center multi-Region replication to include the Region in which your Omni space is hosted.

If you do not connect an identity provider, your team can sign in to Omni using an IAM user or IAM role (for example, through a deep link from the CloudWatch console).

Share the domain with your team

Domain setup is complete. Member accounts can now create their own spaces. Each space is automatically linked to this domain. Share the domain URL (`https://<domain-name>.cloudwatch-omni.global.app.aws`) with account owners who need to create spaces.

Part 2: Create a space in a member account

This section is for **account owners** in member accounts. You create a space in your account and Region; the space is automatically linked to the Organization domain.

Only **one space** can exist in a given account and Region combination.

Prerequisites

- An Organization domain must already exist for your AWS Organization. If it does not, ask your Organization administrator to create one (see Part 1).
- Permission to create a space. Grant this to administrators only.
- The AWS Region you want to work in. A space belongs to one Region, and your IAM Identity Center instance must be in that same Region. If your IAM Identity Center instance is in a different Region, enable IAM Identity Center multi-Region replication to include your desired space Region.

Create a space

1. Open the Omni setup page in the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/home#settings:/omni.
2. Choose **Create space**.
3. Enter a space name and confirm the account and Region.
4. Complete the setup. Omni enables the capabilities described in the following section for your account and Region.

Capabilities enabled when you create a space

Setting up a space turns on the following CloudWatch capabilities in your account and Region. These are telemetry features, not permissions. For the IAM roles that space setup creates, see the "IAM roles created during space setup" section of this page.

Capability	What it does	Learn more
CloudWatch Dataset integration	Enables forwarding of your logs and traces to the CloudWatch Dataset, which makes them queryable and correlated together in Omni. You do not manage a separate store or change	*CloudWatch Omni*

Capability	What it does	Learn more
	how you manage existing CloudWatch resources such as log groups.	
OTel metrics enrichment	Enriches AWS metrics with resource attributes in OpenTelemetry format and makes them queryable with PromQL.	CloudWatch OTel metrics enrichment
OTel span ingestion	Enables trace and span ingestion so Omni can receive and correlate distributed traces.	Enable CloudWatch transaction search

These capabilities apply to telemetry received after you complete space setup. Enabling these capabilities does not incur charges on its own. Costs begin when telemetry flows through them. See Amazon CloudWatch pricing.

On the first enablement, Omni backfills your space with up to 7 days of existing CloudWatch logs and traces. Telemetry older than 7 days, and data encrypted with a customer managed AWS KMS key, are not included in the backfill.

IAM roles created during space setup

Space setup creates the following IAM roles in your account. Omni uses these roles to read and correlate your telemetry:

- `CloudWatchOmniOperatorRole` — Omni assumes this role to manage your space, query telemetry, and configure integrations.
- `CloudWatchOmniDatasetIntegrationExecutionRole` — CloudWatch assumes this role to make your telemetry available in Omni through the Dataset.
- `AgentCoreEvaluationRole` — Amazon Bedrock AgentCore assumes this role to run online AI agent evaluations on your telemetry. This role is optional: during setup you keep the default role, choose an existing role, or opt out. See the section called “Online evaluations”.

Centralize telemetry across accounts and Regions

If your workloads span several AWS accounts, you can aggregate telemetry from those accounts into the account and Region that hosts your space. After centralization, you can monitor and investigate your telemetry across all of the source accounts and Regions from the same space.

You aggregate telemetry by creating CloudWatch centralization rules. A centralization rule specifies the **source accounts** and **source Regions** to collect from, and the **destination account** and Region to replicate into. Set the destination to the account and Region that hosts your space. Centralization rules require AWS Organizations and must be created from the management account or by a delegated administrator.

You can create centralization rules before or after you create the space. Centralized telemetry is available in the space after both the space and the rule are active.

For setup instructions, see Cross-account cross-Region log centralization and Cross-account cross-Region metrics centralization.

What centralization adds to your space

- **Logs and metrics from all source accounts and Regions in one place** — Analyze and act on telemetry across all your accounts and Regions from the centralized account. See the section called “Explore and query your telemetry” and the section called “Alerts”.
- **Application map spans accounts** — The application map includes services discovered from all source accounts (when Context Graph is enabled as part of your centralization rules), so you see cross-account dependencies. See the section called “Monitor applications with the application map”.

What to know

- **Centralization does not backfill** — Only telemetry generated after the centralization rule is active is replicated to the destination.
- **Cross-space queries are not supported** — Each space sees the telemetry centralized into its own account and Region. If you need to query across accounts, centralize their telemetry into one account rather than creating separate spaces.
- **Trace centralization requires Transaction Search enablement for each source account** — To centralize traces, enable Transaction Search in all your source accounts. Trace centralization then happens automatically when logs centralization is enabled.

- **Costs** — Centralization replicates telemetry into the destination account. Standard CloudWatch pricing applies to the replicated data. See Amazon CloudWatch pricing.

Configure encryption (optional)

By default, Omni manages the encryption of the data it stores for your space. To control encryption with your own key, configure a customer managed AWS KMS key on the space (when you create the space, or later by updating the space's encryption configuration).

The key must meet these requirements:

- A symmetric encryption key (`ENCRYPT_DECRYPT` key usage), and enabled.
- In the same AWS account and Region as the space.
- The identity attaching the key must have `kms:Decrypt` and `kms:GenerateDataKey` permission on it. Both are checked when you attach the key, so a key granting only one is rejected at attach time rather than failing later on reads or writes. The error names the missing permission.

Data Omni stores for your space, such as dashboards, is encrypted under the key.

The key's own key policy must also allow CloudWatch Omni to use the key. For the statements the service needs and a complete sample key policy, see the section called “Data encryption in CloudWatch Omni”.

Next steps

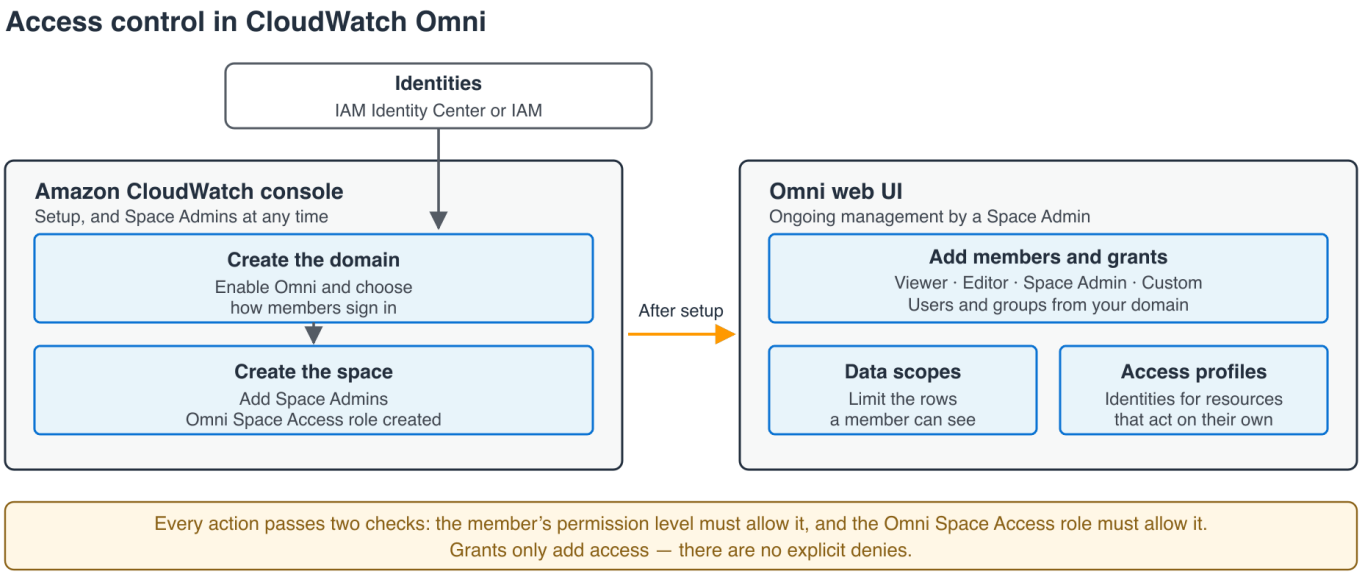
- **Send telemetry** from your applications or agents. See the section called “Send telemetry to CloudWatch Omni”.
- **Add your team** and choose what each person can do. See the section called “Control access to your space”.

Control access to your space

Omni separates setting up access from managing it. You enable Omni, create your domain and space, and grant the first Space Admins in the Amazon CloudWatch console — the setup that the section called “Set up Omni” covers. The identity that creates the space becomes its first Space Admin automatically: setup adds a system managed Space Admin grant for the IAM role or user

that ran the setup flow. That grant cannot be removed while it is the last Space Admin grant. Space Admins are the one part of access you can keep managing in the CloudWatch console: it has a page for adding and removing them at any time after setup. Everything else is in the Omni web UI (the other permission levels, data scopes, and access profiles), where a Space Admin adds members and each member's grants decide what they can do and see. This page explains the model; the pages under it carry the procedures and the reference.

The following diagram shows where each piece is created. The **Omni Space Access role** in it is the IAM role the space assumes to reach your AWS resources; the next section explains how it works alongside permission levels.



Grants and permission levels

When you add a member to a space or edit their access, you create a **grant** that carries a **permission level**. A grant belongs to a **principal**: a user, a group, an IAM user or role, an access profile, or an alert. This guide calls them all principals, and calls the people among them members.

Each grant carries one permission level:

- **Viewer**: Read-only access to the whole space, including queries and the Omni agent.
- **Editor**: Create, modify, and delete resources other than integrations, run queries, and manage datasets, in addition to everything a Viewer can do.
- **Space Admin**: Full access, including managing members and their permissions, and connecting, changing, or removing integrations.

- **Custom**: Exactly the actions you list, optionally limited to specific resources. See the section called “Custom grant actions”.

A member can hold grants directly and through groups, and the grants combine: their effective access is the union of what all the grants allow. For the tiered levels the highest one wins, and a Custom grant adds exactly the actions it lists. There are no explicit denies; a grant only ever adds access.

A grant applies everywhere the member works (the Omni web UI, the IDE extension, and the API), and one permission level governs every kind of resource: threads, dashboards, alerts, integrations, and agents. You manage grants in the Omni web UI. For the procedure, see the section called “Manage space members and permissions”.

Two layers decide every action

Two conditions must hold for an action to succeed in a space:

- The **member's permission level** decides whether that person is allowed to request the operation.
- The **Omni Space Access role** decides whether the space itself is permitted to carry out the operation against your AWS resources. This is the IAM role you configure for the space when you set it up. The space assumes this role to read your telemetry and, for evaluations, to invoke models on your behalf.

A member can perform an operation only when their permission level allows it and the Space Access role grants the underlying AWS permission it depends on. Raising a member to Space Admin does not add permissions that the Space Access role does not have.

Use permission levels to decide what each person on your team can do, and keep the Space Access role scoped to the AWS permissions the space needs. For how to configure the role during setup, see the section called “Set up Omni”.

Who can sign in and be added

Identities come from your domain, not from the space. Sign-in is configured through AWS IAM Identity Center at the domain, so members use the identities you already manage, including an external provider such as Okta or Microsoft Entra ID connected through IAM Identity Center. A domain without an identity provider uses IAM instead: members sign in with an IAM user or IAM role, for example through a deep link from the CloudWatch console.

Grants attach to those identities. The users and groups you can add as members are the ones your domain's identity provider supplies, and the IAM users and roles of the space's account. To configure the identity provider, see the section called “Set up Omni”.

Limit what members can see

By default, every member can read all of the space's telemetry. A **data scope** on a grant narrows that: it filters log and trace rows at read time, wherever the member reads — queries, dashboards, traces, and the Omni agent's answers.

A data scope is a read-time filter on those rows, not a containment boundary for the space. It does not filter metrics, and it does not extend to dataset creation and export, to AI-assisted summaries, or to prompt playground runs, so a member who is scoped out of a field in queries can still reach that content by building a dataset over the underlying traces. Scope the dataset, export, and AI-feature actions as well. See the section called “Limit what members can see” and the section called “Security best practices for CloudWatch Omni”.

Resources that act on their own

Some resources act when no one is signed in: an alert evaluates its query on a schedule. An **access profile** gives such a resource an identity of its own, with its own grants. See the section called “Access profiles”.

Where to go next

To do this	See
Add members and manage their grants	the section called “Manage space members and permissions”
Narrow which telemetry rows a member sees	the section called “Limit what members can see”
Look up the actions a Custom grant can carry	the section called “Custom grant actions”
Give an alert or other resource its own identity	the section called “Access profiles”
Review the security model end to end	the section called “Security in CloudWatch Omni”

Manage space members and permissions

Add members to your space, give them grants, and remove access when it is no longer needed. Only a Space Admin can add, edit, or delete grants. For what grants and permission levels are and how they combine, see the section called “Control access to your space”.

Available in the Omni web UI only.

Prerequisites

- You are signed in to your space in the Omni web UI.
- You hold a grant with the **Space Admin** permission level.
- The people you want to add can sign in to your domain. The users and groups available depend on how your domain's identity provider is set up; see the section called “Set up Omni”.

Add a member

1. Open **Settings**, and under **Space** choose **Manage permissions**. Choose **Add grants**. To change an existing grant instead, choose the principal's name to open their grants, open the grant, and choose **Edit grant**.
2. Under **Principals**, search for users, groups, access profiles, IAM roles, or alerts, and select one or more. (When editing an existing member, the member is already selected.)
3. Under **Grant name**, enter a name that identifies this grant, such as `dashboard-read`.
4. Under **Permission level**, choose **Viewer**, **Editor**, **Space Admin**, or **Custom**. For what each level allows, see the section called “Control access to your space”. For a custom grant, select the actions to allow and, optionally, the resources they apply to; the actions are listed in the section called “Custom grant actions”.
5. (Optional) To limit the data this grant can see, under **Advanced: limit the data this principal can see**, choose **Add limits to data visibility** and build the data scope. For what a scope can express, see the section called “Limit what members can see”.
6. Choose **Add**, or **Save changes** when editing. The principal appears in the table with the new grant listed beside it.

Remove access

To remove access, choose the principal's name to open their grants, select the grant, and choose **Delete**. Deleting a grant cannot be undone. When you delete a member's last grant, they immediately lose access to every resource in the space.

- A space must always keep at least one Space Admin, so you cannot delete the last grant that provides Space Admin access.
- You cannot edit or delete your own grants. Ask another Space Admin to change your access.
- An IAM Identity Center user also holds every grant given to a group they belong to, and deleting their own grants does not remove that inherited access. To remove all of their access, remove the user from those groups in IAM Identity Center or your identity provider, or delete the group's grants, which affects every member of the group.

System managed grants

Some grants are created by the service. The table marks them as system managed, and they cannot be edited or deleted.

Give a resource its own access

Some resources act when no one is signed in. An alert, for example, evaluates its query on a schedule. An access profile gives such a resource an identity of its own, and you add a profile as a member and give it a permission level as you would a person, with one exception: **Space Admin** is not available to an access profile, because space administration is reserved for people. See the section called “Access profiles”.

Limit what members can see

By default, every member of the space can read all of its telemetry. To give a member a narrower view, add a **data scope** to their grant. A data scope is a set of conditions that a row of telemetry must match for that member to see it. Rows that do not match are filtered out at read time, before results reach the member. The stored data is not changed. To control what telemetry is captured or stored in the first place, see the section called “Protect sensitive data”.

A data scope is enforced on the query path. It applies wherever the member reads telemetry from the CloudWatch Dataset, including query results, dashboards, trace and session views, and the answers the Omni agent gives from its own queries. The same scope applies in the Omni web UI, the IDE extension, and the API.

> **Note**
>
> A data scope applies only to telemetry read from the CloudWatch Dataset. It doesn't filter investigation results from AWS DevOps Agent. It also doesn't filter other data from outside the Dataset, such as CloudWatch Logs log groups or third-party tools.
> If your space is connected to an AWS DevOps Agent space, a member with a data scope can still see investigation content. This content can come from CloudWatch Logs log groups outside their scope. For more information, see the section called "Investigate with the AWS DevOps Agent".

You configure a data scope on the grant, under `Advanced: limit the data this principal can see`, when you add or edit a member (step 5 of the procedure in the section called "Manage space members and permissions").

Fine-grained access control with a data scope

A data scope has two parts: the signal types the conditions apply to, and one or more condition groups.

Signal types. Under `Applies to`, choose `Logs`, `Traces`, or both. The condition groups filter only the signal types you select. A signal type you do not select is not filtered, and the member sees all of its rows.

Condition groups. Each group is a set of conditions on the fields of a row. You can add conditions of four kinds:

- **By agent** matches rows whose resource attribute `service.name` is one of the agents you select and whose `aws.service.type` is `gen_ai_agent`. Use this to give a team access to only the agents it owns.
- **By service** matches rows whose resource attribute `service.name` is one of the services you select. This matches every instrumented service, agents included.
- **By account** matches rows whose resource attribute `cloud.account.id` is one of the AWS accounts you select. Use this when several accounts send telemetry to one space and each team owns an account.
- **By field value** matches rows where a field you choose is any one of the values you enter. You can use any field that appears in your telemetry, such as a deployment environment, a tenant

identifier, or a custom attribute your application sets. The field picker lists fields discovered from your telemetry, and you can also type a field name.

A condition matches when the field is equal to one of the values you list. Within one group, every condition must match (AND). Across groups, a row is visible when it matches any group (OR). A grant with no condition groups applies no data scope, and the member sees everything.

When a member holds more than one grant, their view is the union of what each grant allows. A grant without a data scope gives the member the full view, whatever scopes their other grants carry. To keep a member's view narrow, put a data scope on every grant you give them, including grants they inherit through a group.

You can set a data scope on any permission level, including Space Admin. It limits what the member can read. It does not limit the space-management actions the level allows.

Common patterns

Goal	Data scope
A team sees only the agents it owns	One group. By agent, listing the team's agents.
Each team sees only telemetry from its own AWS account	One grant per team. One group, By account, listing that team's account ID.
An on-call engineer sees production traces for one service	Applies to Traces. One group with By service for the service and By field value for the environment attribute set to the production value.
A reviewer sees two unrelated slices	Two groups, one for each slice. The member sees rows that match either group.

What a data scope does and does not filter

A data scope decides which rows a member can see. It does not hide fields inside a row. A member who can see a span or a log record sees every field on it, including prompt and response content. To keep specific content out of every member's view, redact it before it reaches the space. See the section called “Protect sensitive data”.

Aggregates follow the scope. A count, an error rate, or a latency percentile is computed over the rows the member can see. Two members with different scopes can get different numbers from the same query.

> **Note**
>
> The By agent, By service, and By account pickers suggest values found in your recent telemetry. You can also type a value that has not been seen yet, for example an agent you are about to deploy. A condition on a value that never appears in the data matches no rows.

Data scope in the grant JSON

The `JSON` view of the add-grants form shows the grant definition, and you can edit it directly. You can also reuse it with the API, the AWS CDK, or the AWS CLI. It carries the grant name, permission level, action scopes, and tags. The principal comes from the `Name` picker and is not part of the JSON. A data scope is an action scope on a `DataSet` resource. This example limits a Viewer to traces from two agents, or to any trace whose deployment environment is `prod`. Logs stay unrestricted: the scope names only `TRACES` in `signalTypes`, so the `READ` permission level keeps its default full access to logs.

```
{
  "name": "checkout-team-viewer",
  "permission": "READ",
  "scopedActions": [
    {
      "actions": [
        "cloudwatch:GetRecords",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricData"
      ],
      "resources": [
        {
          "resourceType": "DataSet",
          "signalTypes": ["TRACES"],
          "rowScopeGroups": [
            [
              {
                "operator": "IN",
                "field": "resource['attributes']['service.name']",
                "values": ["checkout-agent", "returns-agent"]
```

```
              },
              {
                "operator": "IN",
                "field": "resource['attributes']['aws.service.type']",
                "values": ["gen_ai_agent"]
              }
            ],
            [
              {
                "operator": "IN",
                "field": "resource['attributes']['deployment.environment']",
                "values": ["prod"]
              }
            ]
          ]
        }
      ]
    }
  ]
}
```

`rowScopeGroups` is a list of groups. Conditions inside a group combine with AND, and groups combine with OR. `IN` is the only operator. A nested attribute is written in bracket notation, so a dotted attribute name such as `service.name` is read as one name.

The example above uses a single group. This one uses two groups, each with two conditions, so it shows both axes at once. It limits a Viewer to traces from the `checkout-agent` **in `prod`**, or to any trace from the `returns-service` **in account 111122223333**. Logs stay unrestricted, since `signalTypes` names only `TRACES`.

```
{
  "name": "returns-reviewer",
  "permission": "READ",
  "scopedActions": [
    {
      "actions": [
        "cloudwatch:GetRecords",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricData"
      ],
      "resources": [
        {
```

```
          "resourceType": "DataSet",
          "signalTypes": ["TRACES"],
          "rowScopeGroups": [
            [
              {
                "operator": "IN",
                "field": "resource['attributes']['service.name']",
                "values": ["checkout-agent"]
              },
              {
                "operator": "IN",
                "field": "resource['attributes']['deployment.environment']",
                "values": ["prod"]
              }
            ],
            [
              {
                "operator": "IN",
                "field": "resource['attributes']['service.name']",
                "values": ["returns-service"]
              },
              {
                "operator": "IN",
                "field": "resource['attributes']['cloud.account.id']",
                "values": ["111122223333"]
              }
            ]
          ]
        }
      ]
    }
  ]
}
```

A trace is visible when it matches group 1 (`service.name` is `checkout-agent` **and** `deployment.environment` is `prod`) or group 2 (`service.name` is `returns-service` **and** `cloud.account.id` is `111122223333`). Every condition inside a group must match (AND); a row that matches any group is visible (OR).

Custom grant actions

A Custom grant allows exactly the actions you list. Nothing is added by default. Each action is named `service:Action`, such as `cloudwatch:GetAlert`. When you choose `Custom` in the add-

grants form, the `Select actions` drawer groups the actions of each service into `Read`, `Write`, and `Permissions management`, the same way the IAM console does. Use this page to decide which actions a member needs for a task, and to see what each action allows. For the add-grants procedure, see the section called “Manage space members and permissions”.

A member who holds a Custom grant and a Viewer, Editor, or Space Admin grant gets the combination of both. To keep a member's access narrow, give them only the Custom grant.

Limits on Custom grants

- **Resource and tag scoping applies only to CloudWatch actions.** You can limit a `cloudwatch:` action to named resources, to resources with a tag, or to a data scope only when the action acts on a resource type: telemetry query actions (a `DataSet` data scope), alerts, dashboards, access profiles, and integrations. Other `cloudwatch:` actions, such as `cloudwatch:GetSpace`, apply to the whole space and accept no scope. Actions from every other service, such as `bedrock-agentcore:` or `logs:`, accept no resource or tag scope. They apply to everything the Omni Space Access role can reach.
- **One service per action group.** An action group in the drawer holds actions from one service. To grant actions from two services, add two groups.
- **Actions from other services also need the Space Access role.** A member can use a `bedrock-agentcore:` or `logs:` action only when the Omni Space Access role for the space also allows it. See the section called “Control access to your space”. Model inference is separate. The base policy lets a space set up and view evaluations, but invoking a model also requires the optional AWS managed policy `CloudWatchOmniModelInferencePolicy` on the role.
- **DevOps Agent actions are not part of the Space Access role.** To use the DevOps Agent from a space, attach the AWS managed policy `AIDevOpsAgentFullAccess` to the role.

Actions by service

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:StartTelemetryQuerySession` , `cloudwatc`	Run queries in the explorer, in dashboards, and in the alert query preview: open and	Members who read telemetry. Included in Viewer.

Service	Actions	What the actions are used for	Grant it to
	h:ListTelemetryQuerySessions ,cloudwatch:StopTelemetryQuerySession ,cloudwatch:StopTelemetryQuery ,cloudwatch:GetTelemetryQueryResults	close query sessions, cancel a query, and read its results. Each is checked under its own name. The data a query reads is checked separately, as the next two rows describe.	

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:GetRecords`	Read logs and traces. GetRecords is the permission behind `StartTelemetryQuery` and `ListTelemetryFields` : Omni checks it for every logs or traces dataset a query references, and for the dataset that `ListTelemetryFields` names. A dataset named without a `logs.` or `traces.` prefix, including `default`, counts as both and is checked once.	Members who read logs and traces. Included in Viewer.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:GetMetricData`, `cloudwatch:ListMetrics`	Read metrics. When `StartTelemetryQuery` references a `metrics.` dataset, that part is checked as both actions. `ListTelemetryFields` on a metrics dataset is checked as `cloudwatch:ListMetrics` alone. A query that spans logs or traces and metrics needs this row and the `cloudwatch:GetRecords` row.	Members who read metrics. Included in Viewer.
CloudWatch	`cloudwatch:GetContextGraph`	See the application map.	Members who read telemetry. Included in Viewer.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:CreateOmniThread ,cloudwatch:GetOmniThread , cloudwatch:UpdateOmniThread ,cloudwatch:ListOmniThreads , cloudwatch:DeleteOmniThread ,cloudwatch:StartOmniThreadSession`	Ask the Omni agent questions and manage conversations. The agent reads with the member's own access, so grant the query actions too.	Members who use the Omni agent. Included in Viewer.
CloudWatch	`cloudwatch:SubmitFeedback`	Rate an answer or send feedback.	Members who use the Omni agent. Included in Editor.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:ListAlerts`, `cloudwatch:GetAlert`, `cloudwatch:CreateAlert`, `cloudwatch:UpdateAlert`, `cloudwatch:DeleteAlert`	See alerts and their state, and create, change, or delete them. Creating an alert also needs `cloudwatch:ListAccessProfiles` and `cloudwatch:AssumeAccessProfile` on the profile the alert uses. Can be limited to named alerts or tagged alerts.	Members who work with alerts. Viewer includes the List and Get actions. Editor includes all five.
CloudWatch	`cloudwatch:ListOmniDashboards`, `cloudwatch:GetOmniDashboard`, `cloudwatch:CreateOmniDashboard`, `cloudwatch:UpdateOmniDashboard`, `cloudwatch:DeleteOmniDashboard`	See dashboards, and create, change, or delete them. Viewing a dashboard also needs the query actions. Can be limited to named or tagged dashboards.	Members who work with dashboards. Viewer includes the List and Get actions. Editor includes all five.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:ListViews`, `cloudwatch:GetView`, `cloudwatch:CreateView`, `cloudwatch:UpdateView`, `cloudwatch:DeleteView`	See saved views, and create, change, or delete them.	Members who work with saved views. Viewer includes the List and Get actions. Editor includes all five.
CloudWatch	`cloudwatch:GetSpace`, `cloudwatch:ListSpaces`, `cloudwatch:UpdateSpace`	Read the space's details, list the spaces the member can open, and change the space's name and configuration.	Viewer includes `cloudwatch:GetSpace` and `cloudwatch:ListSpaces`. Space Admin includes `cloudwatch:UpdateSpace`.
CloudWatch	`cloudwatch:GetPreferences`, `cloudwatch:UpdatePreferences`	Read and save the member's own preferences.	Any member. Included in Viewer.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:ListAccessGrants`, `cloudwatch:GetAccessGrant`, `cloudwatch:CreateAccessGrant`, `cloudwatch:DeleteAccessGrant`	Read the grants in the space, and add or remove them.	Viewer includes the List and Get actions. Space Admin includes `cloudwatch:CreateAccessGrant` and `cloudwatch:DeleteAccessGrant`.
CloudWatch	`cloudwatch:ListAccessProfiles`, `cloudwatch:GetAccessProfile`, `cloudwatch:CreateAccessProfile`, `cloudwatch:UpdateAccessProfile`, `cloudwatch:DeleteAccessProfile`, `cloudwatch:AssumeAccessProfile`	See access profiles, manage them, and act as one. `cloudwatch:AssumeAccessProfile` is what a member needs to set a profile on an alert, and what an alert needs to run. Can be limited to named or tagged profiles.	Viewer includes the List and Get actions. Editor adds `cloudwatch:AssumeAccessProfile`. Space Admin includes all six.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:ListIntegrations`, `cloudwatch:GetIntegration`, `cloudwatch:CreateIntegration`, `cloudwatch:UpdateIntegration`, `cloudwatch:DeleteIntegration`	You can see, connect, change, or remove integrations such as Slack and AWS DevOps Agent. You can limit these actions to named or tagged integrations.	Viewer includes the List and Get actions. Space Admin includes all five.
CloudWatch	`cloudwatch:InvokeIntegration`	Send a message through an integration. Can be limited to named integrations and, for Slack, to channels.	Access profiles that alerts use to notify. No permission level includes it.
CloudWatch	`cloudwatch:GetIntelligenceConfiguration`, `cloudwatch:PutIntelligenceConfiguration`	Read the account's intelligence configuration, and turn intelligence features on or off.	Viewer includes `cloudwatch:GetIntelligenceConfiguration`. Editor includes `cloudwatch:PutIntelligenceConfiguration`.

Service	Actions	What the actions are used for	Grant it to
CloudWatch	`cloudwatch:TagResource` , `cloudwatch:UntagResource` , `cloudwatch:ListTagsForResource`	Read, add, and remove tags on Omni resources. Only these resource types accept tags: spaces, domains, organization domains, access grants, organization access grants, access profiles, dashboards, alerts, views, integrations, and datasets. The Space Access role denies tagging of any other CloudWatch resource. These are the tags that a tag-scoped grant matches on.	Members who organize resources by tag, and Space Admins who set up tag-scoped grants.

Service	Actions	What the actions are used for	Grant it to
Bedrock AgentCore	`bedrock-agentcore:ListEvaluators`, `bedrock-agentcore:GetEvaluator`, `bedrock-agentcore:CreateEvaluator`, `bedrock-agentcore:UpdateEvaluator`, `bedrock-agentcore:DeleteEvaluator`, `bedrock-agentcore:Evaluate`	See, author, and run evaluators for agent evaluations.	Members who run or author agent evaluations.

Service	Actions	What the actions are used for	Grant it to
Bedrock AgentCore	bedrock-agentcore:ListDatasets ,bedrock-agentcore:GetDataset ,bedrock-agentcore:ListDatasetVersions ,bedrock-agentcore:ListDatasetExamples ,bedrock-agentcore:CreateDataset ,bedrock-agentcore:UpdateDataset ,bedrock-agentcore:CreateDatasetVersion ,bedrock-agentcore:AddDatasetExamples ,bedrock-agentcore:UpdateDatasetExamples ,bedrock-	See and manage the datasets that evaluations run against. The tag actions also read or change tags on evaluators and online evaluation configs.	Members who author agent evaluations.

Service	Actions	What the actions are used for	Grant it to
	agentcore:DeleteDatasetExamples ,bedrock-agentcore:DeleteDataset ,bedrock-agentcore:TagResource ,bedrock-agentcore:UntagResource ,bedrock-agentcore:ListTagsForResource		

Service	Actions	What the actions are used for	Grant it to
Bedrock AgentCore	`bedrock-agentcore:ListOnlineEvaluationConfigs` ,`bedrock-agentcore:GetOnlineEvaluationConfig` , `bedrock-agentcore:CreateOnlineEvaluationConfig` ,`bedrock-agentcore:UpdateOnlineEvaluationConfig` ,`bedrock-agentcore:DeleteOnlineEvaluationConfig`	Set up online evaluations that run on live traffic without a user.	Members who set up online evaluations.
Bedrock	`bedrock:ListFoundationModels` ,`bedrock:ListInferenceProfiles`	Choose a model when creating an evaluator.	Members who author agent evaluations.

Service	Actions	What the actions are used for	Grant it to
Bedrock	`bedrock:InvokeModel` , `bedrock:InvokeModelWithResponseStream` , `bedrock-mantle:CreateInference` , `bedrock-mantle:CallWithBearerToken`	Invoke the model when an evaluation or the playground runs. The `bedrock-mantle` actions cover models served through the `bedrock-mantle` endpoint in Amazon Bedrock. The Space Access role must also carry the optional `CloudWatchOmniModelInferencePolicy` .	Members who run evaluations or use the playground.
IAM	`iam:PassRole`	Pass a service role in your account to CloudWatch or to Bedrock AgentCore, for example the role that online evaluations run as.	Members who set up the space or online evaluations.

Service	Actions	What the actions are used for	Grant it to
IAM	`iam:ListRoles` ,`iam:GetRole` ,`iam:ListAttachedRolePolicies` ,`iam:ListUsers` ,`iam:ListUserTags` ,`iam:ListServiceSpecificCredentials`	Fill the role picker, inspect an existing role during Azure VM ingestion setup, and list the ingestion users and their OTLP bearer tokens.	Members who set up telemetry sources or choose service roles.
IAM	`iam:CreateServiceLinkedRole`	Create the AWS Config service-linked role that Azure topology needs.	Members who set up Azure topology.
CloudWatch Logs	`logs:CreateLogGroup` ,`logs:CreateLogStream` ,`logs:PutLogEvents`	Write evaluation results to log groups.	Members who run evaluations.
CloudWatch Logs	`logs:DescribeLogGroups`	List log groups for the online evaluation log-group picker.	Members who set up online evaluations.
CloudWatch Logs	`logs:DescribeIndexPolicies` ,`logs:PutIndexPolicy`	Index the service name on the spans log group so online evaluations can sample by service.	Members who set up online evaluations.

Service	Actions	What the actions are used for	Grant it to
KMS	`kms:DescribeKey` , `kms:GenerateDataKey` , `kms:Decrypt`	Validate and use a customer managed key for space encryption and for stored provider keys.	Members who configure encryption or store provider keys.
Lambda	`lambda:InvokeFunction` , `lambda:GetFunction`	Run code-based custom evaluators.	Members who use code evaluators.
Secrets Manager	`secretsmanager:ListSecrets` , `secretsmanager:CreateSecret` , `secretsmanager:TagResource` , `secretsmanager:PutSecretValue` , `secretsmanager:GetSecretValue` , `secretsmanager:DeleteSecret`	Store, read, rotate, and delete model provider API keys used by evaluators.	Members who add provider keys.

Service	Actions	What the actions are used for	Grant it to
Config	`config:PutServiceLinkedConfigurationRecorder`, `config:DeleteServiceLinkedConfigurationRecorder`, `config:DescribeConfigurationRecorder`, `config:DescribeConfigurationRecorderStatus`	Turn the AWS Config recorder on or off, and read its status, to build the Azure topology map.	Members who set up Azure topology.

Access profiles

An access profile is an identity for a resource that acts when no one is signed in. An alert, for example, evaluates its query on a schedule. The profile holds the permissions the resource uses at that time.

A profile is reusable. You create it once, grant it permissions, and choose which resources may assume it, so several alerts can share one profile instead of each carrying a copy of the same permissions.

> **Note**
>
> Access profiles and permission levels solve different problems. A permission level governs a **person** and applies wherever they work. A profile governs a **resource** acting on its own. See the section called “Control access to your space”.

What an access profile contains

Every profile has two halves:

Half	What it answers
Permissions	What this profile is allowed to do: the grants it holds, and the integrations it can invoke.
Who can assume	Which principals are allowed to act as this profile. In the console, you choose alerts: either every alert in the space or specific ones.

Neither half works alone. A profile with permissions that nothing is allowed to assume cannot be used. A profile that resources can assume but that has no permissions authorizes nothing. The console catches this when you create an alert, but a profile that loses its permissions later fails without an error. See the "Alerts and profiles" section of this page.

A profile belongs to one space, and its name must be unique within that space.

System managed access profiles

Creating a space also creates one system managed access profile:

Profile	Purpose
DefaultAccessProfileForAsyncWorkflows	The identity the Omni agent uses when it investigates a Slack mention and no one is signed in. Only the space's Omni agent can assume it. It starts with no permissions, so Slack mention investigations cannot read your data until you grant it Viewer access. See the section called “Connect Slack”.

A system managed profile is created with the space and maintained by the service. You cannot rename, edit, or delete it. You can still grant it permissions, the same way you grant any profile. Profiles that you create are customer managed, and you control them fully.

Create a profile for the resources that need one, scoped to the job it does. For example, an alert that reads one service's telemetry and posts to one Slack channel gets a profile that can do only that.

Create an access profile

1. In the console, open **Settings**, and under **Space** choose **Access Profiles**.
2. Choose **Create profile**.
3. Enter a **Name**. This is what you select from later, so name it for the job it does, such as `Read-only observability`. Add an optional **Description** and **Tags**.
4. Under **Permissions**, leave **Allow alerts that use this profile to read telemetry and send notifications** selected to give the profile the actions that alert evaluation needs. Clear it to start with an empty profile and grant permissions yourself.
5. Under **Who can assume**, choose which alerts may act as this profile. **Allow all alerts** is selected by default. Clear it to pick specific alerts.
6. Choose **Create profile**.

The profile appears in the list with its name and description. To show its ID as well, use the column settings. Open it to see whether *you* can assume it: the detail page reports **You are allowed to assume this access profile** or **You are not allowed to assume this access profile**.

Grant a profile its permissions

The **Permissions** checkbox on the create form gives a new profile the actions that alert evaluation needs. The console creates this grant for you with exactly these actions:

- Query session and results: `cloudwatch:StartTelemetryQuerySession`, `cloudwatch:ListTelemetryQuerySessions`, `cloudwatch:StopTelemetryQuerySession`, `cloudwatch:StopTelemetryQuery`, `cloudwatch:GetTelemetryQueryResults`
- Telemetry reads: `cloudwatch:GetRecords`, `cloudwatch:GetMetricData`, `cloudwatch:ListMetrics`
- Notifications: `cloudwatch:InvokeIntegration`

For what each action allows, see the section called “Custom grant actions”.

To grant anything beyond that, grant it the way you grant a person access: open the profile and choose **Add grant**, or on the **Manage permissions** page add the profile as you would a member. See the section called “Manage space members and permissions”.

A profile's detail page lists every grant it holds in the same grants table as the **Manage permissions** page, and marks the grants the service created as system managed. Those cannot be edited or deleted.

Alerts themselves hold only one kind of grant. When you add a grant for an alert, or for **ALL** alerts, the form locks the grant to the single `cloudwatch:AssumeAccessProfile` action and asks you to pick the access profile it applies to. All of the alert's other permissions come from the profile it assumes.

Two limits apply to whatever you grant:

- **A profile cannot administer the space.** Space administration is reserved for people. A resource acting as a profile can never add members, change permissions, or manage profiles, no matter how it is granted.
- **The Omni Space Access role is the ceiling.** A profile can hold only permissions that the role also holds, and it never adds to them. Both layers apply together, exactly as they do for a member's permission level. The role is created during setup (see the section called “Set up Omni”) and its policies are described in the section called “Identity and access management for CloudWatch Omni”.

A profile can hold **Viewer**, **Editor**, or **Custom** grants. **Space Admin** is not offered for a profile.

Alerts and profiles

Every alert carries a profile. When you create an alert in the console, the **Access profile** field is preselected to the profile in your space with the most grants. Change it if the alert should run with narrower access. The query preview on the create form runs with your own access. When the alert runs, it reads with the profile's access, so results can differ.

An alert evaluates its query *as* its profile. If alerts cannot assume the profile, or the profile cannot read the data the query names, the alert is not created and the console offers a **Manage access profiles** link. This check runs only when you create or update the alert. If the profile later loses read access to the data the query uses, the alert returns no rows and reports **no data** rather than an error.

Two conditions must hold:

- **Both you and the alert must be allowed to assume the profile.** Open the profile to check your own access. Its page reports whether you are allowed to assume it. Leaving **Allow all alerts** selected covers the alert side. If you narrowed a profile to specific alerts, a new alert is not among them until you add it.
- **The profile must be able to read the telemetry.** A profile that can invoke integrations but cannot read your data delivers nothing. Leaving the **Permissions** checkbox selected when you create the profile covers the telemetry reads and the notification call.

If an alert has reported no data since you created it, check the profile before you check the query. See the section called “Alerts”.

Review and manage a profile

Open a profile from the list to see:

- **Whether you can assume it.** You need this to set the profile on an alert.
- **Description** and **Tags**.
- **Its grants.** Every grant the profile holds, in the same table as **Manage permissions**, with its name, ID, permission level, and whether it is system managed. Open a grant to read its JSON, edit it, or delete it. **Add grant** adds one. A system managed grant is read-only.

The profile page does not list who can assume it. To see or change which alerts, users, or groups may act as a profile, open **Manage permissions**, choose the principal (for alerts, the **ALL** row or a specific alert), and review its `cloudwatch:AssumeAccessProfile` grants there.

On the detail page, **Edit access profile** changes the profile's name, description, and tags. From the list, **Actions** offers **Edit** and **Delete**. Deleting a profile does not check whether alerts still use it. An alert whose profile is deleted can no longer evaluate, so move those alerts to another profile or delete them first. A system managed profile cannot be edited or deleted. If you need different permissions for it, add or remove its grants instead.

Next steps

- To understand how access to a space works, see the section called “Control access to your space”.
- To add members and set their permission levels, see the section called “Manage space members and permissions”.

- To create an alert that uses a profile, see the section called “Alerts”.
- To connect the integrations a profile can invoke, see the section called “Connect Slack”.

AWS managed policies for CloudWatch Omni

CloudWatch Omni provides AWS managed policies for the space operator role, for the domain access role of an organization domain, and for the roles that support organization-level Context Graph enablement. Each policy's details — the description, what it attaches to, the version history, and the complete JSON policy document — are published in the AWS Managed Policy Reference, which is generated from the policy itself and updated automatically when a policy changes. The following table links to each policy's page there.

For guidance on which policy you need for which task, which permissions read across your whole account, and the permissions you provide yourself, see the section called “IAM policies to use CloudWatch Omni”.

The policies

Policy	What it grants
CloudWatchOmniSpaceAccessPolicy	The base policy, always attached to the space operator role. The space experience (space and access-grant management, telemetry queries, the context graph, alerts, dashboard s, access profiles, views, threads, and integrati ons), together with the management side of evaluation.
CloudWatchOmniModelInferencePolicy	Model inference only. Attached only to spaces that use the prompt playground or run evaluations that invoke a model.
CloudWatchOmniDomainAccessPolicy	Domain management for an organization domain. Attached to the domain access role, which CloudWatch Omni assumes to operate the domain. The role is created in the management account during organization

Policy	What it grants
	domain setup. Single-account domains do not use it.
CloudWatchOmniAWSIntegrationPolicy	Resource discovery for Context Graph: lets CloudWatch Omni discover resources and their relationships in an account. Optional on the space operator role — you choose it during space setup. In an organization Centralization rule, CloudWatch Omni also attaches it to the integration roles it provisions in each source account.
AWSCloudWatchOmniServiceRolePolicy	Attached to the `AWSServiceRoleForCloudWatch_Omni` service-linked role, which provisions the integration roles and the AWS Config recorder in member accounts. You cannot attach it to your own identities.

CloudWatch Omni updates to AWS managed policies

The following table describes updates to CloudWatch Omni's AWS managed policies.

Change	Description	Date
CloudWatchOmniSpaceAccessPolicy — Updated policy	CloudWatch Omni added the `cloudwatch:GetProfileRecords` permission so you can query continuous profiling data in a space. Like the other space permissions, it requires the `cloudwatch:HasAccessGrant` condition.	November 18, 2026

Change	Description	Date
CloudWatchOmniSpaceAccessPolicy — Updated policy	CloudWatch Omni added IAM Identity Center read permissions, and `kms:Decrypt` for directories encrypted with a customer managed key, so you can look up users and groups when you grant access to a space. It also added `sns:ListTopics` so you can choose an Amazon SNS topic as an alert notification target.	October 14, 2026
AWSCloudWatchOmniServiceRolePolicy — Updated policy	CloudWatch Omni added the `iam:GetRole` permission. This lets the service-linked role read and reuse the integration role it provisions in an account, instead of recreating it.	September 24, 2026
CloudWatchOmniSpaceAccessPolicy — New policy	CloudWatch Omni added a policy that grants the space experience — space and access-grant management, telemetry queries, alerts, dashboards, access profiles, views, threads, and integrations — together with the management side of evaluation.	September 22, 2026

Change	Description	Date
CloudWatchOmniModelInferencePolicy — New policy	CloudWatch Omni added a policy that grants model inference for the prompt playground and for evaluations that invoke a model.	September 22, 2026
CloudWatchOmniDomainAccessPolicy — New policy	CloudWatch Omni added a policy that grants domain management for an organization domain, attached to the domain access role created during organization domain setup.	September 22, 2026
CloudWatchOmniAWSIntegrationPolicy — New policy	CloudWatch Omni added a policy that grants resource discovery for Context Graph. It is optional on the space operator role, and CloudWatch Omni attaches it to the integration roles it provisions in the source accounts of an organization Centralization rule.	September 22, 2026
AWSCloudWatchOmniServiceRolePolicy — New policy	CloudWatch Omni added a policy for the `AWSServiceRoleForCloudWatch_Omni` service-linked role, which provisions the integration roles and the AWS Config recorder in member accounts.	September 22, 2026

Change	Description	Date
CloudWatch Omni started tracking changes	CloudWatch Omni started tracking changes for its AWS managed policies.	September 22, 2026

Send telemetry to CloudWatch Omni

CloudWatch Omni reads what Amazon CloudWatch has ingested. Configure your application, OpenTelemetry exporter, or collector to send telemetry to a CloudWatch OTLP endpoint. After that telemetry is in the CloudWatch Dataset for your space, it is available in CloudWatch Omni.

Where to start

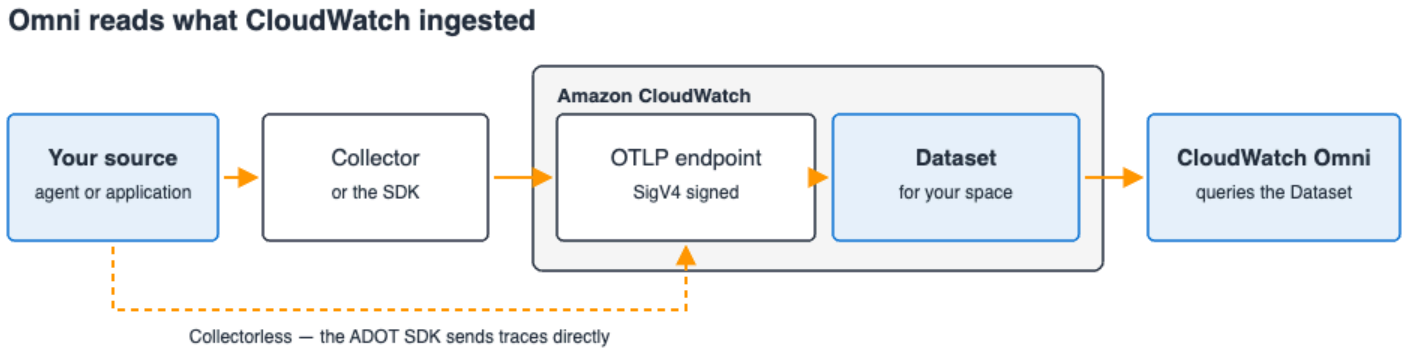
- **Send application telemetry to CloudWatch Omni**
  - **Environment:** Amazon EC2, Amazon ECS, Amazon EKS, Azure VM, Azure AKS, Google Compute Engine, or Google Kubernetes Engine.
  - On AWS Lambda there is no agent to deploy. See Send application telemetry from AWS Lambda.
- **Send AI agent telemetry to CloudWatch Omni**
  - **Environment:** Amazon Bedrock AgentCore, AWS Lambda, Amazon EC2, Amazon ECS, or Amazon EKS. The links open Step 2; Step 1 is the same for every environment.

Your situation	Start here
My application already emits OpenTelemetry and sends it to a collector I run	Nothing to install. Configure your collector to sign requests and send to an endpoint below (see **Bring your own collector** under "Choose how you collect"), then confirm data arrives with the **Your application appears** check in Step 3 of Send application telemetry.
My application already emits OpenTelemetry but has no collector	Step 1 of Send application telemetry, **Deploy the CloudWatch agent**, then Step 2, **Point**

Your situation	Start here
	your application at the agent, and Step 3, **Verify**. Skip Step 4.
My application is not instrumented yet	Send application telemetry to CloudWatch Omni, all four steps.
Telemetry stopped arriving, or never arrived	The troubleshooting table on the page you set up with: applications, Lambda, or AI agents.
My workload runs somewhere not listed	Any OpenTelemetry collector or SDK can send to the endpoints below with SigV4-signed requests. See the collection comparison in Getting started with OTLP in the Amazon CloudWatch User Guide.

> **Note**
>
> **Assisted setup.** The **Add source** wizard in the Omni web UI (**Settings › Ingestion**) walks through the same choices and generates commands for your selections. From your coding agent, the Omni skills in the Agent Toolkit for AWS do the same; see the section called "Use Omni skills". From your editor, see the section called "Develop with the IDE extension".

How telemetry reaches your space



Forwarding to the Dataset

The **CloudWatch Dataset integration** is the step between Amazon CloudWatch and the Dataset in the diagram above: it forwards your CloudWatch logs and traces into the Dataset for your space. Setting up a space in the CloudWatch console also creates the integration and an execution role that the CloudWatch Logs service assumes, so forwarding starts without a separate step. The space and the integration are separate resources — console setup creates them together, and you manage them independently. For the roles setup creates, see the section called “Set up Omni for a single account”. The integration forwards logs and traces together. Metrics are not forwarded; Omni queries them directly from CloudWatch Metrics.

There is one Dataset integration per account in a Region, and forwarding is same-account and same-Region. The execution role's `logs:IntegrateWithDataset` permissions control which log groups forward. Log groups in the Delivery log class do not forward, regardless of the role's permissions.

To manage the integration directly — for example, with the AWS CLI or infrastructure as code — follow these three steps.

Step 1. Create an execution role for the CloudWatch Logs service to assume. This example uses the name `CloudWatchOmniDatasetIntegrationRole`; you can use any name. The role must be in the same account as the Dataset and the integration — IAM does not support passing a role across accounts. Use this trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": { "aws:SourceAccount": "<account-id>" },
      "ArnLike": { "aws:SourceArn": "arn:aws:observabilityadmin:<region>:<account-
id>:dataset-integration/default" }
    }
  }]
}
```

In both `Condition` values, `<account-id>` is the account that owns the role, the Dataset, and the integration. With any other account ID the create call still succeeds, but no logs forward.

Step 2. Attach this permissions policy to the role. The `log-group:*` resource forwards all log groups in the account except Delivery-class log groups; to forward only specific log groups, replace * with a log group name. If your space's Dataset is encrypted with a customer managed AWS KMS key, the key's policy must also allow this role `kms:Encrypt`, `kms:Decrypt`, and `kms:GenerateDataKey` on records entering the Dataset — see the `AllowCallerDatasetForwarding` statement in the sample key policy in the section called “Data encryption in CloudWatch Omni”.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "logs:IntegrateWithDataset",
      "Resource": ["arn:aws:logs:<region>:<account-id>:log-group:*"]
    },
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutRecords",
      "Resource": "*"
    }
  ]
}
```

Step 3. Create the Dataset integration, passing the execution role. Creating or updating the integration requires `iam:PassRole` on the execution role.

```
aws observabilityadmin create-dataset-integration \
    --role-arn arn:aws:iam::<account-id>:role/CloudWatchOmniDatasetIntegrationRole \
    --region <region>
```

Changing what is forwarded. Updates to the execution role's `logs:IntegrateWithDataset` permissions take effect within a few minutes. Changes to what is forwarded apply going forward: removing a log group stops future records from forwarding, and records already in the Dataset remain until they pass their retention period. To change the execution role, use `update-dataset-integration`; to stop all forwarding, use `delete-dataset-integration`.

Choose how you collect

The collection path you choose affects which features you get: entity correlation and Container Insights are not available on every path. For a comparison of the CloudWatch agent, an upstream OpenTelemetry Collector, a custom collector, and collectorless export with the AWS Distro for OpenTelemetry (ADOT) SDK, see Getting started in the Amazon CloudWatch User Guide.

The CloudWatch agent is the recommended path for applications, and it is the path Send application telemetry to CloudWatch Omni uses. AI agents and Lambda functions do not use a collector: an AI agent exports its traces directly from the ADOT SDK, and Lambda delivers segments and logs itself. To configure the CloudWatch agent for OTLP, see Amazon CloudWatch agent in the Amazon CloudWatch User Guide.

Bring your own collector. Point a self-managed OpenTelemetry Collector's exporters at the endpoints below. The collector's role needs the permissions named in the "Authentication" section of this page, and each exporter needs SigV4 signing configured; for the collector configuration, see Getting started with OTLP in the Amazon CloudWatch User Guide.

Choose an endpoint

Endpoint	Use for	Authentication
Traces	Application traces forwarded by the CloudWatch agent, and AgentCore runtime traces. The primary signal for Omni.	SigV4, or a bearer token
Metrics	Custom OpenTelemetry metrics, queryable with PromQL.	SigV4, or a bearer token
Logs	Log records, forwarded to a log group and log stream you name in request headers.	SigV4, or a bearer token

For the hostnames, per-signal request headers, protocol support, and the ingestion limits that apply to each, see OTLP endpoints.

> **Important**
>
> Enable Transaction Search in the Region before you send traces. It is an account-level setting, and until it is enabled spans do not arrive.

Authentication

AWS Signature Version 4 (SigV4) with short-lived credentials is the default for every path. A workload running on AWS compute signs with the role it already has, but that role must carry the permissions for the path you use, and the three paths do not share one permission set. Each procedure page shows the statement to attach.

- Collector path: the CloudWatch agent's managed policy, `CloudWatchAgentServerPolicy`.
- An AgentCore runtime: the trace-delivery actions `xray:GetSamplingRules`, `xray:GetSamplingTargets`, `xray:PutTelemetryRecords`, and `xray:PutTraceSegments` on the runtime's execution role.

Workloads outside AWS can federate for temporary credentials; the Azure and Google Cloud procedures do this. Restrict bearer tokens to scenarios where short-lived credentials are not feasible. The metrics, logs, and traces endpoints accept a long-lived bearer token, with a separate API key for each service. See Setting up bearer token authentication for Metrics, Setting up bearer token authentication for Logs, and Setting up bearer token authentication for traces. Token creation requires IAM mutating actions, so it is an administrator task.

Environment support

For the environments, languages, and frameworks with a documented path, the signals each produces, and the minimum library versions, see Supported environments, languages, and frameworks.

On-premises hosts can obtain temporary AWS credentials with IAM Roles Anywhere, so a collector on your own hosts signs requests with SigV4 instead of storing a long-lived key.

Next steps

When telemetry is arriving, see the section called “Monitor AI agents” and the section called “Monitor applications with the application map”. To control who can access the space, see the

section called “Control access to your space”. To keep sensitive content out of the telemetry you send and store, see the section called “Protect sensitive data”.

Send application telemetry

Deploy the CloudWatch agent where your application runs: Amazon EC2, Amazon ECS, Amazon EKS, an Azure VM, Azure Kubernetes Service, a Google Compute Engine instance, or Google Kubernetes Engine. Then point your application's OpenTelemetry exporter at the agent, and confirm that data arrives in your space. If your application is not instrumented yet, add OpenTelemetry for Python, Node.js, Java, .NET, or Go in the last step. The agent receives OpenTelemetry Protocol (OTLP) data from your application and forwards metrics, traces, and logs to CloudWatch.

If you are monitoring an AI agent, see Send AI agent telemetry. If your application runs on AWS Lambda, see Send application telemetry from AWS Lambda; Lambda has no collector to deploy.

Steps 1 and 2 show one environment at a time: expand the section for where your application runs in each step. Sections are collapsed so you can see the whole page; open only yours.

If your application already uses OpenTelemetry, do Steps 1 to 3 and stop; Step 4 is only for applications that are not instrumented yet. If you already run your own OpenTelemetry collector or the ADOT SDK, you need only an endpoint: see "Choose an endpoint" in Send telemetry to CloudWatch Omni.

> **Note**
>
> **Assisted setup.** The **Add source** wizard in the Omni web UI (**Settings › Ingestion**) generates these commands for your selections. From your coding agent, the Omni skills in the Agent Toolkit for AWS do the same; see the section called “Use Omni skills”. The rest of this page is the manual path.

Prerequisites

- **Transaction Search is enabled** in the account and Region that receive traces. Traces sent before it is enabled are not searchable. See Transaction Search.
- You can attach IAM policies to the role your compute uses: an instance role, a task role, the agent's service account role, or the federated role on Azure or Google Cloud.

- For Step 4, your application's language is one of Python, Node.js, Java, .NET, or Go. For more information about the frameworks with documented notes, see Supported environments, languages, and frameworks. Other frameworks still work with the custom-instrumentation snippets.

Step 1: Deploy the CloudWatch agent (platform team or administrator)

Expand the environment your application runs on. Each section gives the install commands and the same resources as infrastructure as code, as tabs.

Amazon EC2

On Amazon EC2, you attach a managed policy to the instance role, install the agent with the OTLP preset (per instance, or fleet-wide with a State Manager association), then continue to Step 2.

Grant permissions

The instance role needs `CloudWatchAgentServerPolicy`, which covers metrics, logs, and traces, bound to the instance through an instance profile.

Install the agent and enable OTLP

Install the agent

```
# 1. Connect (SSH, or Session Manager — no key or open port needed)
aws ssm start-session --target instance-id

# 2. Install the CloudWatch agent from Amazon S3
curl -fsSL -o /tmp/amazon-cloudwatch-agent.rpm \
  https://amazoncloudwatch-agent.s3.amazonaws.com/amazon_linux/amd64/latest/amazon-
cloudwatch-agent.rpm
sudo rpm -Uvh /tmp/amazon-cloudwatch-agent.rpm

# 3. Enable OTLP
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
  -a fetch-config -c default:otel -s
```

Replace *instance-id* with the ID of the Amazon EC2 instance you want to connect to.

- Install version 1.300071.0 or later, the first with OTLP support. The agent is available from Amazon S3, Systems Manager, and Amazon ECR, but not from the Amazon Linux yum repository.

- `default:otel` turns on the OTLP receiver plus host metrics and span metrics.

> **Important**
>
> A manual install does not scale. New or auto-replaced instances launch without a collector. For any Auto Scaling group, keep the agent installed with a State Manager association (in the CDK, Terraform, and CloudFormation tabs), EC2 user data, or a golden AMI.

CDK TypeScript

```
import { Role, ServicePrincipal, ManagedPolicy, InstanceProfile } from 'aws-cdk-lib/
aws-iam';
import { CfnAssociation } from 'aws-cdk-lib/aws-ssm';

const role = new Role(this, 'OmniEC2TelemetryRole', {
  assumedBy: new ServicePrincipal('ec2.amazonaws.com'),
  managedPolicies: [
    ManagedPolicy.fromAwsManagedPolicyName('CloudWatchAgentServerPolicy'),
  ],
});
new InstanceProfile(this, 'OmniEC2TelemetryProfile', { role });
new CfnAssociation(this, 'OmniCollectorAssociation', {
  name: 'AWS-ConfigureAWSPackage',
  associationName: 'omni-cloudwatch-agent',
  targets: [{ key: 'tag:omni:monitor', values: ['true'] }],
  parameters: { action: ['Install'], name: ['AmazonCloudWatchAgent'] },
  scheduleExpression: 'rate(30 days)',
});
```

CDK Python

With the AWS CDK in Python or Java, build the same resources: an IAM role assumable by `ec2.amazonaws.com` with the managed policy `CloudWatchAgentServerPolicy`, an instance profile bound to that role, and a `CfnAssociation` with the same properties as the CloudFormation example.

Terraform

```
resource "aws_ssm_association" "omni_collector" {
```

```
  name             = "AWS-ConfigureAWSPackage"
  association_name = "omni-cloudwatch-agent"
  targets {
    key    = "tag:omni:monitor"
    values = ["true"]
  }
  parameters = { action = "Install", name = "AmazonCloudWatchAgent" }
  schedule_expression = "rate(30 days)"
}

resource "aws_instance" "app" {
  # …
  iam_instance_profile = aws_iam_instance_profile.app.name
  user_data = <<-EOF
    #!/bin/bash
    T=$(curl -sS -X PUT http://169.254.169.254/latest/api/token -H "X-aws-ec2-
metadata-token-ttl-seconds: 600")
    until curl -sf -H "X-aws-ec2-metadata-token: $T" http://169.254.169.254/latest/
meta-data/iam/security-credentials/ | grep -q .; do sleep 5; done
    curl -fsSL -o /tmp/cwa.rpm https://amazoncloudwatch-agent.s3.amazonaws.com/
amazon_linux/amd64/latest/amazon-cloudwatch-agent.rpm
    rpm -Uvh /tmp/cwa.rpm
    /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config
 -c default:otel -s
    systemctl enable amazon-cloudwatch-agent
  EOF
}
```

- An instance profile created in the same apply can take a minute to reach the instance metadata service. The user data waits for the role before `fetch-config`, because an agent that starts without credentials never sends telemetry.

CloudFormation

```
Resources:
  OmniEC2TelemetryRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement: [{ Effect: Allow, Action: sts:AssumeRole, Principal: { Service:
 ec2.amazonaws.com } }]
      ManagedPolicyArns:
```

```
        - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
  OmniEC2TelemetryProfile:
    Type: AWS::IAM::InstanceProfile
    Properties: { Roles: [!Ref OmniEC2TelemetryRole] }
  OmniCollectorAssociation:
    Type: AWS::SSM::Association
    Properties:
      Name: AWS-ConfigureAWSPackage
      AssociationName: omni-cloudwatch-agent
      Targets: [{ Key: tag:omni:monitor, Values: ["true"] }]
      Parameters: { action: [Install], name: [AmazonCloudWatchAgent] }
      ScheduleExpression: rate(30 days)
```

For the full agent configuration reference, see Amazon CloudWatch agent in the Amazon CloudWatch User Guide.

Amazon ECS

On Amazon ECS (Fargate), you attach a managed policy to the task role, add the agent as a second container in the same task with the application depending on it, then continue to Step 2. Your application sends OTLP to `localhost`; the task shares one network namespace.

Grant permissions and egress

The task role needs `CloudWatchAgentServerPolicy`; the execution role needs `AmazonECSTaskExecutionRolePolicy`. The task needs egress to pull the agent image and reach the OTLP endpoints: a public IP in a public subnet, a NAT gateway, or VPC endpoints.

> **Important**
>
> The application container must depend on the agent container with condition `START`. The agent image has no health check, so condition `HEALTHY` hangs startup.

Add the sidecar to the task definition

Setup script

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/
scripts/aws/setup.sh | \
  CWAGENT_PLATFORM=aws_ecs \
```

```
  CWAGENT_AWS_REGION=<region> \
  CWAGENT_AWS_ENABLE_TRANSACTION_SEARCH=true \
  sh
```

- The script creates the task role and prints the agent container definition and the task role ARN. Paste both into your task definition.
- It defaults to the Fargate launch type. For the EC2 launch type, set `CWAGENT_AWS_ECS_LAUNCH_TYPE=ec2`; the printed definition then adds host port mappings and container links.

CDK TypeScript

```
const agent = taskDef.addContainer('cwagent', {
  image: ecs.ContainerImage.fromRegistry('public.ecr.aws/cloudwatch-agent/
cloudwatch-agent:latest'),
  environment: { CW_CONFIG_CONTENT: JSON.stringify({ opentelemetry: { collect:
 { otlp: {} } } }) },
  logging: ecs.LogDrivers.awsLogs({ streamPrefix: 'cwagent' }),
});
app.addContainerDependencies({ container: agent, condition:
 ecs.ContainerDependencyCondition.START });
taskDef.taskRole.addManagedPolicy(ManagedPolicy.fromAwsManagedPolicyName('CloudWatchAgentSer
```

CDK Python

```
agent = task_def.add_container("cwagent",
    image=ecs.ContainerImage.from_registry("public.ecr.aws/cloudwatch-agent/
cloudwatch-agent:latest"),
    environment={"CW_CONFIG_CONTENT": json.dumps({"opentelemetry": {"collect":
 {"otlp": {}}}})},
    logging=ecs.LogDrivers.aws_logs(stream_prefix="cwagent"))
app.add_container_dependencies(
    ecs.ContainerDependency(container=agent,
 condition=ecs.ContainerDependencyCondition.START))
task_def.task_role.add_managed_policy(iam.ManagedPolicy.from_aws_managed_policy_name("CloudW
```

With the AWS CDK (Java), add a second container with image `public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest` and environment variable `CW_CONFIG_CONTENT` set to `{"opentelemetry":{"collect":{"otlp":{}}}}`, make the application container depend on it with condition `START`, and attach `CloudWatchAgentServerPolicy` to the task role.

Terraform

```
container_definitions = jsonencode([
  { name = "cwagent", image = "public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest",
    environment = [{ name = "CW_CONFIG_CONTENT", value = "{\"opentelemetry\":
{\"collect\":{\"otlp\":{}}}}" }],
    logConfiguration = { logDriver = "awslogs", options = { "awslogs-group" = "/ecs/
cwagent", "awslogs-region" = "<region>", "awslogs-stream-prefix" = "cwagent" } } },
  { name = "app", image = "<your-image>",
    dependsOn = [{ containerName = "cwagent", condition = "START" }] }
])
```

CloudFormation

```
ContainerDefinitions:
  - Name: cwagent
    Image: public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest
    Environment:
      - { Name: CW_CONFIG_CONTENT, Value: '{"opentelemetry":{"collect":{"otlp":
{}}}}' }
  - Name: app
    Image: <your-image>
    DependsOn: [{ ContainerName: cwagent, Condition: START }]
```

For the agent configuration reference, see Amazon CloudWatch agent.

Amazon EKS

On Amazon EKS, you install the CloudWatch Observability add-on with OTLP enabled and give its agent credentials, then continue to Step 2. The add-on runs the agent as a DaemonSet and exposes a `cloudwatch-agent` Service in the `amazon-cloudwatch` namespace; your pods send OTLP to that Service over cluster DNS, not to `localhost`. The add-on is installed with the AWS CLI regardless of how you deploy your application, so there is one installation path here.

```
curl -fsSL https://raw.githubusercontent.com/aws/amazon-cloudwatch-agent/main/scripts/
aws/setup.sh | \
  CWAGENT_PLATFORM=aws_eks \
  CWAGENT_K8S_CLUSTER_NAME=<my-cluster> \
  CWAGENT_AWS_REGION=<region> \
  CWAGENT_AWS_ENABLE_TRANSACTION_SEARCH=true \
```

```
  sh
```

- The script installs the EKS Pod Identity Agent, creates the agent role with a Pod Identity association, and installs the add-on with OTLP enabled. Pod Identity supplies the agent's credentials, so you do not annotate the service account for IAM roles for service accounts (IRSA).

Three settings must be right or telemetry never arrives, with no error in the agent log:

- The OTLP receiver must be bound to `0.0.0.0`, not the default `localhost`, or it refuses cross-pod traffic. If application OTLP is refused after the script runs, apply the override and restart the DaemonSet:

```
aws eks update-addon --cluster-name <my-cluster> --addon-name amazon-cloudwatch-
observability \
  --configuration-values '{"agent":{"config":{"opentelemetry":{"collect":{"otlp":
{"grpc_endpoint":"0.0.0.0:4317","http_endpoint":"0.0.0.0:4318"}}}}}}'
kubectl -n amazon-cloudwatch rollout restart daemonset/cloudwatch-agent
```

- The agent must have credentials through Pod Identity (the script) or an IRSA-annotated service account. The add-on's `--service-account-role-arn` flag does not annotate the service account; without the annotation the agent falls back to the node role and its exports are rejected.
- Application logs are in the cluster-scoped log group `/aws/cwagent/<my-cluster>/otlp`, not in `/aws/cwagent/otlp`.

For a manual install with IRSA and the add-on configuration reference, see Install the CloudWatch Observability add-on in the Amazon CloudWatch User Guide.

Azure VM

On an Azure VM, you federate the VM's managed identity to an IAM role and install the CloudWatch agent on the VM, then continue to Step 2. The agent sends telemetry with temporary AWS credentials instead of a stored key; your application sends OTLP to `localhost`.

The complete procedure, including the OpenID Connect (OIDC) federation, the IAM role trust policy, and the onboarding script, is in Install the CloudWatch agent on Azure in the Amazon CloudWatch User Guide. Follow the VM section there, then return to Step 2.

Azure AKS

On Azure Kubernetes Service, you federate workload identity for the `cloudwatch-agent` service account to an IAM role and install the CloudWatch Observability Helm chart, then continue to Step 2. Pods send OTLP to the `cloudwatch-agent` Service over cluster DNS.

The complete procedure, including the OIDC federation and the Helm values, is in Install the CloudWatch agent on Azure in the Amazon CloudWatch User Guide. Follow the AKS section there, then return to Step 2.

> **Note**
>
> Unlike the EKS add-on, the AKS Helm chart's OTLP receiver accepts cross-pod traffic by default; no `0.0.0.0` override is needed.

Google Compute Engine

On a Google Compute Engine instance, you federate the instance's service account to an IAM role and install the CloudWatch agent on the instance, then continue to Step 2. The agent sends telemetry with temporary AWS credentials instead of a stored key; your application sends OTLP to `localhost`.

The complete procedure, including the service account federation, the IAM role trust policy, and the onboarding scripts, is in Install the CloudWatch agent on Google Cloud in the Amazon CloudWatch User Guide. Follow the Google Compute Engine (GCE) section there, then return to Step 2.

Google Kubernetes Engine

On Google Kubernetes Engine, you federate the cluster's OIDC issuer for the `cloudwatch-agent` service account to an IAM role and install the CloudWatch Observability Helm chart, then continue to Step 2. Pods send OTLP to the `cloudwatch-agent` Service over cluster DNS.

The complete procedure, including the OIDC federation and the Helm values, is in Install the CloudWatch agent on Google Cloud in the Amazon CloudWatch User Guide. Follow the Google Kubernetes Engine (GKE) section there, then return to Step 2.

Step 2: Point your application at the agent (developer)

Set these environment variables on the application, never on the agent: one agent serves many services and already enriches telemetry with infrastructure attributes. The endpoint and the

place the variables live depend on your environment, and the port and protocol depend on your language; expand the same section you chose in Step 1, then choose your language.

Amazon EC2

On Amazon EC2, set these in the application's service unit or start script, for example `Environment=` lines in a systemd unit:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Amazon ECS

On Amazon ECS, set these in the application container's `environment` block:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Amazon EKS

On Amazon EKS, set these in the container's `env` block in the pod spec:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Azure VM

On an Azure VM, set these in the application's service unit or start script:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
```

```
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Azure AKS

On Azure Kubernetes Service, set these in the container's `env` block in the pod spec:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Google Compute Engine

On a Google Compute Engine instance, set these in the application's service unit or start script:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Google Kubernetes Engine

On Google Kubernetes Engine, set these in the container's `env` block in the pod spec:

Python, Java, .NET, Go

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Node.js

```
OTEL_SERVICE_NAME=<checkout-api>
OTEL_RESOURCE_ATTRIBUTES=service.namespace=<checkout>
OTEL_EXPORTER_OTLP_ENDPOINT=http://cloudwatch-agent.amazon-
cloudwatch.svc.cluster.local:4317
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_METRICS_EXPORTER=otlp
OTEL_LOGS_EXPORTER=otlp
```

Step 3: Verify

Use the automated verification in the Omni web UI. In your space, open **Settings › Ingestion** and choose your source: the **Add source** flow's verification step confirms the agent is reporting and your application's telemetry is arriving, and points at the failing setting when it is not. Run it after every deployment change.

If verification fails, see the "Troubleshoot missing telemetry" section of this page.

Verify manually

The agent is reporting

Amazon EC2

On Amazon EC2, `sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status` reports `running`. In your space, host metrics for the instance appear within about five minutes.

Amazon ECS

On Amazon ECS, the task shows both containers `RUNNING`. In your space, container metrics for the task appear within about five minutes.

Amazon EKS

On Amazon EKS, `kubectl -n amazon-cloudwatch get daemonset cloudwatch-agent` shows all pods ready. In your space, container metrics for the cluster appear within about five minutes.

Azure VM

On an Azure VM, `amazon-cloudwatch-agent-ctl -a status` reports `running`. In your space, host metrics for the VM appear within about five minutes.

Azure AKS

On Azure Kubernetes Service, `kubectl -n amazon-cloudwatch get daemonset cloudwatch-agent` shows all pods ready. In your space, container metrics for the cluster appear within about five minutes.

Google Compute Engine

On a Google Compute Engine instance, `sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status` reports `running`. In your space, host metrics for the instance appear within about five minutes.

Google Kubernetes Engine

On Google Kubernetes Engine, `kubectl -n amazon-cloudwatch get daemonset cloudwatch-agent` shows all pods ready. In your space, container metrics for the cluster appear within about five minutes.

Your application appears

If you run your own OpenTelemetry Collector instead of the CloudWatch agent, this is your check. If your application is not instrumented yet, do Step 4 first, then return to this check. Restart the application, then open your space. In **Application map**, the service appears under the `service.name` you set. In **Trace Explorer**, a trace for a recent request shows the entry span for your service with the resource attributes you set. Application logs are in `/aws/cwagent/otlp` on EC2, ECS, Azure VMs, and Compute Engine instances, or `/aws/cwagent/<cluster>/otlp` on EKS, AKS, and GKE. A custom metric is queryable with PromQL by its OpenTelemetry name, for example `sum({__name__="orders.placed"})`.

Step 4 (optional): Instrument your application with OpenTelemetry (developer)

Choose your language. The choice applies to every code sample in this step and is remembered on the other pages in this guide. For Python, Node.js, Java, and .NET, most HTTP frameworks, database clients, and messaging libraries are traced without code changes. Go has no auto-instrumentation agent, so initialize the SDK in code and wrap your handlers with the library instrumentation for your framework. The table in each tab lists the frameworks with documented notes, and the OpenTelemetry registry lists the rest. For a framework not covered, the custom-instrumentation snippet still works.

Install the OpenTelemetry SDK

Python

```
pip install opentelemetry-distro opentelemetry-exporter-otlp
opentelemetry-bootstrap -a install
```

- In a container, both commands belong in the Dockerfile.
- On a host running Amazon Linux 2023, the default `python3` is 3.9 and the CloudWatch plugin requires Python 3.10 or later: install and use `python3.11` (`dnf install -y python3.11 python3.11-pip`).

Node.js

```
npm install @opentelemetry/sdk-node @opentelemetry/auto-instrumentations-node \
  @opentelemetry/exporter-trace-otlp-grpc @opentelemetry/api @opentelemetry/api-logs
```

Java

```
curl -L -o opentelemetry-javaagent.jar \
  https://github.com/open-telemetry/opentelemetry-java-instrumentation/releases/
latest/download/opentelemetry-javaagent.jar
```

- The agent does not put the OpenTelemetry API on your compile classpath. To add custom instrumentation, add `io.opentelemetry:opentelemetry-api` at compile scope, not `provided`, because the agent bridges to the application's own API classes at runtime.

.NET

```
# Console application
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol

# ASP.NET Core or gRPC service
dotnet add package OpenTelemetry.Extensions.Hosting
dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol
dotnet add package OpenTelemetry.Instrumentation.AspNetCore
dotnet add package OpenTelemetry.Instrumentation.Http
```

Go

```
go get go.opentelemetry.io/otel \
  go.opentelemetry.io/otel/sdk \
  go.opentelemetry.io/otel/sdk/metric \
  go.opentelemetry.io/otel/sdk/log \
  go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracehttp \
```

```
  go.opentelemetry.io/otel/exporters/otlp/otlpmetric/otlpmetrichttp \
  go.opentelemetry.io/otel/exporters/otlp/otlplog/otlploghttp \
  go.opentelemetry.io/contrib/bridges/otelslog \
  github.com/aws-observability/cloudwatch-plugin-otel-go

# Library instrumentation for your framework
go get go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp
  # net/http
go get go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc
  # gRPC
```

Add the CloudWatch plugin for OpenTelemetry with the SDK. The plugin generates the request, error, and duration span metrics that power the service views (including the **Errors** metric), and it meters every span before sampling, so those metrics reflect all requests rather than the sampled subset. It is available for Python, Node.js, Java, .NET, and Go. The following table lists the package names and links. The plugin needs a metrics pipeline to emit into. If your application exports traces but no metrics, the plugin stays inert and **Errors** remains unpopulated. Configure a metrics exporter alongside your trace exporter.

If you instrument with the ADOT SDK instead and sample, your span metrics are computed from the sampled spans only.

Language	Package	Documentation
Java	`software.amazon.opentelemetry:cloudwatch-plugin-otel`	Java plugin README
Python	`cloudwatch-plugin-otel`	Python plugin README
Node.js	`@aws/cloudwatch-plugin-otel`	Node.js plugin README
.NET	`AWS.OpenTelemetry.CloudWatchPluginOtel`	.NET plugin README

Language	Package	Documentation
Go	`github.com/aws-observability/cloudwatch-plugin-otel-go`	Go plugin README

Enable auto-instrumentation

Run your application under the auto-instrumentation agent for its language. Framework, HTTP, and database calls are traced without code changes. Go has no auto-instrumentation agent. For Go, initialize the SDK once at startup instead.

Python

```
opentelemetry-instrument python <app.py>
# FastAPI
opentelemetry-instrument uvicorn <app:app> --host 0.0.0.0 --port 8000
```

- `opentelemetry-instrument` wraps the process, so it replaces your entry point. For FastAPI under uvicorn, wrap `uvicorn`, not `python app.py`.
- For Django, set `DJANGO_SETTINGS_MODULE` in the environment and run with `--noreload`.
- Python exports logs only when `OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true` is also set in the environment.

Node.js

```
node --require @opentelemetry/auto-instrumentations-node/register <app.js>
```

Java

```
java -javaagent:./opentelemetry-javaagent.jar -jar <app.jar>
```

.NET

```
# ASP.NET Core or gRPC: install and enable the profiler
curl -sSfL https://github.com/open-telemetry/opentelemetry-dotnet-instrumentation/
releases/latest/download/otel-dotnet-auto-install.sh -o install.sh && sh install.sh
```

```
. $HOME/.otel-dotnet-auto/instrument.sh
dotnet <MyService.dll>
```

```
// Console application: bootstrap once at startup
using var tracerProvider =
 Sdk.CreateTracerProviderBuilder().AddSource("checkout").AddOtlpExporter().Build();
using var meterProvider  =
 Sdk.CreateMeterProviderBuilder().AddMeter("checkout").AddOtlpExporter().Build();
```

Go

The following Go code example initializes the SDK once at startup. Save it in your application as `otel.go`.

```
package main

import (
    "context"
    "log/slog"

    "github.com/aws-observability/cloudwatch-plugin-otel-go/spanmetrics"
    "go.opentelemetry.io/contrib/bridges/otelslog"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/exporters/otlp/otlplog/otlploghttp"
    "go.opentelemetry.io/otel/exporters/otlp/otlpmetric/otlpmetrichttp"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracehttp"
    sdklog "go.opentelemetry.io/otel/sdk/log"
    sdkmetric "go.opentelemetry.io/otel/sdk/metric"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
)

func setupOTel(ctx context.Context) (func(context.Context) error, error) {
    // service.name and service.namespace come from OTEL_SERVICE_NAME and
 OTEL_RESOURCE_ATTRIBUTES.
    res, err := resource.New(ctx, resource.WithFromEnv())
    if err != nil {
        return nil, err
    }

    me, err := otlpmetrichttp.New(ctx)
    if err != nil {
        return nil, err
```

```
    }
    mp := sdkmetric.NewMeterProvider(
        sdkmetric.WithReader(sdkmetric.NewPeriodicReader(me)),
        sdkmetric.WithResource(res),
    )
    otel.SetMeterProvider(mp)

    te, err := otlptracehttp.New(ctx)
    if err != nil {
        return nil, err
    }
    tp := sdktrace.NewTracerProvider(
        sdktrace.WithResource(res),
        // CloudWatch plugin: meter every span before sampling, into mp.

 sdktrace.WithSampler(spanmetrics.NewAlwaysRecordSampler(spanmetrics.SamplerFromEnv())),
        sdktrace.WithSpanProcessor(spanmetrics.NewSpanMetricsProcessor(mp)),
        sdktrace.WithBatcher(te),
    )
    otel.SetTracerProvider(tp)

    le, err := otlploghttp.New(ctx)
    if err != nil {
        return nil, err
    }
    lp := sdklog.NewLoggerProvider(
        sdklog.WithProcessor(sdklog.NewBatchProcessor(le)),
        sdklog.WithResource(res),
    )
    // Send the default slog logger's records through OpenTelemetry.
    slog.SetDefault(slog.New(otelslog.NewHandler("checkout",
 otelslog.WithLoggerProvider(lp))))

    return func(ctx context.Context) error {
        _ = tp.Shutdown(ctx)
        _ = mp.Shutdown(ctx)
        return lp.Shutdown(ctx)
    }, nil
}
```

- Call `setupOTel(ctx)` at the start of `main()`, before you serve requests, and defer the returned function so the final spans, metrics, and logs are exported on shutdown.

- The setup registers the CloudWatch plugin. `NewAlwaysRecordSampler` wraps the sampler that `OTEL_TRACES_SAMPLER` selects, and `NewSpanMetricsProcessor` records the span metrics into the meter provider, so create the meter provider before the tracer provider.
- Log with the default `slog` logger's context methods, such as `slog.InfoContext`, so each record is exported and correlated with the active span. A logger that you create separately with `slog.New` is not connected to OpenTelemetry.

Add custom instrumentation

Set business attributes on the active span, mark your entry span as a server span so it counts in request metrics, and record a custom metric. These snippets assume the SDK is initialized by the step above.

Python

```
from opentelemetry import trace, metrics
from opentelemetry.trace import SpanKind

tracer = trace.get_tracer("checkout")
orders = metrics.get_meter("checkout").create_counter("orders.placed")

def place_order(cart):
    with tracer.start_as_current_span("place_order", kind=SpanKind.SERVER) as span:
        span.set_attribute("cart.items", len(cart))
        orders.add(1)
```

Framework	Notes
FastAPI	Traced automatically under `opentelemetry-instrument uvicorn .`
Flask	Traced automatically under `opentelemetry-instrument python .`
Django	Views are traced automatically when `DJANGO_SETTINGS_MODULE` is set in the environment; run with `--noreload`.

Node.js

```
const { trace, metrics } = require('@opentelemetry/api');
const { logs } = require('@opentelemetry/api-logs');
const orders = metrics.getMeter('checkout').createCounter('orders.placed');
const logger = logs.getLogger('checkout');

function placeOrder(cart) {
  trace.getActiveSpan()?.setAttribute('cart.items', cart.length);
  orders.add(1);
  logger.emit({ body: 'order placed', attributes: { 'cart.items': cart.length } });
}
```

- Logs must go through the OpenTelemetry logs API, as `logger.emit()` does here. Setting `OTEL_LOGS_EXPORTER=otlp` alone produces no records.

Framework	Notes
Express	Traced automatically by the auto-instrumentations package.
gRPC	Server and client calls are traced automatically.

Java

```
static final LongCounter ORDERS =

 GlobalOpenTelemetry.getMeter("checkout").counterBuilder("orders.placed").build();

@WithSpan(kind = SpanKind.SERVER)
public void placeOrder(List<Item> cart) {
  Span.current().setAttribute("cart.items", cart.size());
  ORDERS.add(1);
}
```

- `@WithSpan` needs `io.opentelemetry.instrumentation:opentelemetry-instrumentation-annotations`, whose version tracks the agent release, not the API version.

Framework	Notes
Spring Boot	Controllers are traced by the Java agent; annotate service methods with `@WithSpan` for business spans.
gRPC	Calls are traced by the Java agent.

.NET

```
static readonly Meter Meter = new("checkout");
static readonly Counter<long> Orders = Meter.CreateCounter<long>("orders.placed");

Activity.Current?.SetTag("cart.items", cart.Count);
Orders.Add(1);
```

- Register the `Meter` on the metrics builder with `AddMeter("checkout")`. Without it the counter is dropped.

Framework	Notes
ASP.NET Core	Requests are traced by the profiler or `AddAspNetCoreInstrumentation()`. Add `builder.Logging.AddOpenTelemetry(o => o.AddOtlpExporter())` for logs.
gRPC	Client instrumentation is a separate prerelease package (`OpenTelemetry.Instrumentation.GrpcNetClient`); `.AddHttpClientInstrumentation()` covers the HTTP transport meanwhile.

Go

```
import (
    "context"
    "log/slog"

    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/trace"
)

var (
    tracer    = otel.Tracer("checkout")
    orders, _ = otel.Meter("checkout").Int64Counter("orders.placed")
)

func placeOrder(ctx context.Context, cart []Item) {
    ctx, span := tracer.Start(ctx, "place_order",
 trace.WithSpanKind(trace.SpanKindServer))
    defer span.End()
    span.SetAttributes(attribute.Int("cart.items", len(cart)))
    orders.Add(ctx, 1)
    slog.InfoContext(ctx, "order placed", "cart.items", len(cart))
}
```

- Pass the request's `context.Context` through your calls. Spans, metrics, and logs that use it are linked to the request's trace.

The following table lists notes for the Go frameworks.

Framework	Notes
net/http	Wrap the handler with `otelhttp.NewHandler(mux, "checkout")`. Each request is traced as a server span.
gRPC	Create the server with `grpc.NewServer(grpc.StatsHandler(otelgrpc.NewServerHandler()))`. Each call is traced as a server span.

Query custom metrics with PromQL by their OpenTelemetry name, for example `sum({__name__="orders.placed"})`. Restart the application and repeat Step 3.

Troubleshoot missing telemetry

Symptom	Cause and fix
Logs and spans arrive, metrics do not	A metric attribute value is longer than 1024 characters; the CloudWatch OTLP metrics endpoint rejects the whole batch with HTTP 400. Trim the attribute. For Java, the usual cause is `process.command_args` on a long classpath: set `OTEL_JAVA_DISABLED_RESOURCE_PROVIDERS=io.opentelemetry.instrumentation.resources.ProcessResourceProvider`.
Nothing arrives; the SDK logs connection errors	Port and protocol do not match: `4317` is gRPC and `4318` is HTTP. Set `OTEL_EXPORTER_OTLP_PROTOCOL` to match the port. For Go, the exporter package sets the protocol, so use port `4318` with the OTLP/HTTP exporters.
Nothing arrives on ECS; the task hangs at startup	The application container depends on the agent with condition `HEALTHY`. Use `START`.
Nothing arrives on ECS; the agent logs `AccessDenied` or times out	The task role lacks `CloudWatchAgentServerPolicy`, or the task has no egress to the OTLP endpoints. Attach the policy; add a public IP, NAT gateway, or VPC endpoints.
Nothing arrives on EKS; no agent errors	The receiver is bound to `localhost`, or the agent service account has no credentials. Apply the `0.0.0.0` override; use Pod Identity or annotate the service account for IRSA, then restart the DaemonSet.

Symptom	Cause and fix
Nothing arrives from a new EC2 instance	The agent started before the instance profile reached the instance metadata service. Wait for the role in user data before `fetch-config`, or restart the agent.
Nothing arrives from an Azure VM or AKS	The federation is not complete: the IAM role trust policy does not match the identity, or the service account is not annotated. Recheck the federation steps in Install the CloudWatch agent on Azure.
Nothing arrives from a Compute Engine instance or GKE	The federation is not complete: the IAM role trust policy does not match the instance's service account unique ID or the cluster's OIDC issuer, or the instance has no service account attached. Recheck the federation steps in Install the CloudWatch agent on Google Cloud.
The service appears under the agent's name	`OTEL_SERVICE_NAME` is set on the agent, not the application. Move it to the application.
A Python service shows one process span and no route spans	`opentelemetry-instrument` wrapped `python` instead of the server. Wrap `uvicorn` or `gunicorn`.
Traces exist but are not searchable	Transaction Search was enabled after the traces were sent. Enable it, then send new traces.
Only some requests appear in the trace list	Transaction Search indexes 1 percent of spans by default for the trace list; the spans themselves are all stored. Raise the indexing percentage in Transaction Search settings.

Send application telemetry from AWS Lambda

Lambda functions send traces and logs to CloudWatch without a collector. Turn on active tracing so Lambda records the service segments, add a trace SDK for spans around your handler logic, and emit structured logs that join their traces.

If your application runs on Amazon EC2, ECS, EKS, or Microsoft Azure, see Send application telemetry. If you are monitoring an AI agent, see Send AI agent telemetry.

Prerequisites

- **Transaction Search is enabled** in the account and Region that receive traces. See Transaction Search.
- You can change the function's configuration and attach policies to its execution role.

Step 1: Enable active tracing

```
aws lambda update-function-configuration --function-name <my-function> --tracing-config
 Mode=Active
aws iam attach-role-policy --role-name <my-function-role> --policy-arn
 arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess
```

- The default `PassThrough` mode records no segments; only `Active` records the `AWS::Lambda` and `AWS::Lambda::Function` service segments, and only for sampled invocations.
- `AWSXRayDaemonWriteAccess` carries the required `xray:PutTraceSegments` and `xray:PutTelemetryRecords`. The Lambda console attaches it when you enable tracing; with infrastructure as code, attach it yourself.

CDK TypeScript

```
import { Function, Runtime, Code, Tracing } from 'aws-cdk-lib/aws-lambda';

new Function(this, 'CheckoutFn', {
  runtime: Runtime.PYTHON_3_12,
  handler: 'app.handler',
  code: Code.fromAsset('lambda'),
  tracing: Tracing.ACTIVE,   // adds AWSXRayDaemonWriteAccess to the role
});
```

Terraform

```
resource "aws_lambda_function" "checkout" {
  function_name = "checkout"
  handler       = "app.handler"
  runtime       = "python3.12"
  role          = aws_iam_role.checkout_exec.arn
  filename      = "lambda.zip"

  tracing_config {
    mode = "Active"
  }
}

resource "aws_iam_role_policy_attachment" "xray" {
  role       = aws_iam_role.checkout_exec.name
  policy_arn = "arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess"
}
```

CloudFormation

```
Resources:
  CheckoutFn:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      # Handler, Runtime, Role, Code …

  # AWS SAM (AWS::Serverless::Function):
  #   Properties:
  #     Tracing: Active
```

CDK Python

Set the function's tracing mode to `ACTIVE` (`aws_lambda.Tracing.ACTIVE` in Python, `Tracing.ACTIVE` in Java). The CDK `Function` construct adds `AWSXRayDaemonWriteAccess` to the execution role when tracing is active.

Step 2: Add the trace SDK

Attach the AWS Distro for OpenTelemetry (ADOT) Lambda layer for your runtime and set `AWS_LAMBDA_EXEC_WRAPPER` so the layer auto-instruments the handler with no code change, then enrich the active span in your code.

> **Important**
>
> The ADOT layer ARN is Region-, runtime-, and version-specific. Look up the current ARN for your Region and runtime in the ADOT Lambda documentation; do not copy one from another example. Attach it the same way you enabled tracing: `--layers` with the CLI, `layers` in CDK and Terraform, `Layers` in CloudFormation and SAM.

Python

```
# Attach the ADOT Python Lambda layer, then set the exec wrapper:
AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument
```

app.py

```
from opentelemetry import trace

def handler(event, context):
    span = trace.get_current_span()
    span.set_attribute("cart.items", len(event.get("items", [])))
    return {"statusCode": 200, "body": "ok"}
```

Node.js

```
# Attach the ADOT JavaScript Lambda layer, then set the exec wrapper:
AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-handler
```

index.js

```
const { trace } = require('@opentelemetry/api');

exports.handler = async (event) => {
  trace.getActiveSpan()?.setAttribute('cart.items', (event.items ?? []).length);
  return { statusCode: 200, body: 'ok' };
```

```
};
```

Java

```
# Attach the ADOT Java Lambda layer, then set the exec wrapper:
AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-handler   # auto-instrumentation agent
# For the SDK-only wrapper, use /opt/otel-proxy-handler
```

Handler.java

```
import io.opentelemetry.api.trace.Span;

public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent event,
 Context context) {
  Span.current().setAttribute("cart.items", event.getBody().length());
  return new APIGatewayProxyResponseEvent().withStatusCode(200);
}
```

.NET

There is no auto-instrumentation exec wrapper layer for .NET; configure the OpenTelemetry .NET SDK in code. Build a TracerProvider with the OTLP exporter at process start, and wrap the handler body in a span.

Function.cs

```
using System.Diagnostics;
using OpenTelemetry;
using OpenTelemetry.Trace;

// Build once, outside the handler, so it is reused across invocations:
static readonly TracerProvider Tp = Sdk.CreateTracerProviderBuilder()
    .AddSource("checkout").AddOtlpExporter().Build();
static readonly ActivitySource Source = new("checkout");

public APIGatewayProxyResponse Handler(APIGatewayProxyRequest req, ILambdaContext
 ctx)
{
    using var activity = Source.StartActivity("place_order");
    activity?.SetTag("cart.items", req.Body?.Length ?? 0);
    return new APIGatewayProxyResponse { StatusCode = 200, Body = "ok" };
}
```

Step 3: Emit structured logs

Structured log example (Python shown; the same fields apply in every language)

```
import json, logging
from opentelemetry import trace

log = logging.getLogger()

def handler(event, context):
    ctx = trace.get_current_span().get_span_context()
    log.info(json.dumps({
        "msg": "order placed",
        "requestId": context.aws_request_id,
        "trace_id": format(ctx.trace_id, "032x"),
        "items": len(event.get("items", [])),
    }))
    return {"statusCode": 200, "body": "ok"}
```

- Lambda delivers standard output and error to the log group `/aws/lambda/<my-function>`.
- Set the function's log format to JSON (`LoggingConfig.LogFormat = JSON`, or **Logging configuration** in the console) so records are structured and you can change the application log level without a code change.
- Include `requestId` and `trace_id` in every record so log lines join their spans.
- `AWSLambdaBasicExecutionRole` covers log delivery.

Step 4: Verify

Invoke the function, then open your space and choose **Trace Explorer**. A trace for the invocation shows the `AWS::Lambda` and `AWS::Lambda::Function` segments with your handler span beneath them, within about five minutes. In **Application map**, the function appears as a service, and its request, error, and duration metrics are span metrics generated from every exported span. They reflect all invocations, not only sampled ones. In the log group `/aws/lambda/<my-function>`, the JSON records carry the `trace_id` of the trace you just viewed.

> **Note**
>
> When a traced caller invokes the function, such as an API Gateway stage with tracing enabled, the function's spans join the caller's trace. When the caller is not traced, no tracing

header arrives and Lambda decides not to sample the invocation, but Transaction Search still captures the invocation's spans, because it is unaffected by the sampled flag in the trace context. Enable tracing on whatever invokes the function to get one end-to-end trace.

Troubleshoot missing telemetry

Symptom	Cause and fix
No traces at all	Tracing mode is `PassThrough` . Set it to `Active` and confirm the execution role has `AWSXRayDaemonWriteAccess` .
Service segments appear, but no handler spans	The layer is not attached, is the wrong Region or runtime, or `AWS_LAMBDA_EXEC_WR APPER` is unset or wrong for the runtime.
Traces are intermittent or isolated	The caller is not traced, so no sampling decision propagates inbound and the function's spans are not connected to an upstream trace. Enable tracing on whatever invokes the function, such as the API Gateway stage; Transaction Search captures the spans either way.
Logs do not join traces	The log format is not JSON, or records omit `trace_id`. Set the log format to JSON and include the trace ID as shown in Step 3.
Traces exist but are not searchable	Transaction Search was enabled after the traces were sent. Enable it, then invoke again.
Only some invocations appear in the trace list	Transaction Search indexes only a sampled subset of spans for the trace list; all spans are stored. Raise the indexing percentage in Transaction Search settings.

Send AI agent telemetry

Add OpenTelemetry tracing to an AI agent written in Python or Node.js with LangGraph, LangChain, Strands Agents, CrewAI, OpenAI Agents, or the Vercel AI SDK, then run it on Amazon Bedrock AgentCore, AWS Lambda, or on your own compute (Amazon EC2, Amazon ECS, or Amazon EKS) so that its traces reach CloudWatch. CloudWatch Omni reads the model calls, tool calls, and orchestration steps from those traces to analyze and evaluate the agent. On AgentCore, the runtime exports the traces for you; on other compute, the agent exports them directly, as shown in Step 2.

If you are monitoring an application or service instead, see Send application telemetry. For the endpoints and authentication that every path uses, see "Choose an endpoint" in Send telemetry to CloudWatch Omni.

Step 2 shows one environment at a time: expand the section for where your agent runs (Amazon Bedrock AgentCore, AWS Lambda, Amazon EC2, Amazon ECS, or Amazon EKS).

Choose how to instrument

Three approaches lead to the same instrumented agent. Choose the first that fits how you work; this page is the manual path.

Approach	What happens	Where to go
AI coding assistant (default in the console)	Your coding agent installs the Omni skills from the Agent Toolkit for AWS, reads your repository, makes the changes on this page for you, and runs the verification.	the section called “Use Omni skills”
IDE extension	The CloudWatch Omni extension for Visual Studio Code and Kiro detects your framework, applies the instrumentation, and shows traces in the editor as you develop.	the section called “Develop with the IDE extension”

Approach	What happens	Where to go
Manually	You install a library, set environment variables where your agent runs, and verify in your space. Every command is on this page.	Continue below.

Prerequisites

- **Transaction Search is enabled** in the account and Region that receive traces. Traces sent before it is enabled are not searchable. See Transaction Search. On AgentCore, the administrator part of Step 2 enables it if it is not.
- Your agent is written in **Python** or **Node.js**, and its framework is in Supported environments, languages, and frameworks.
- On **AgentCore**, you can change the runtime's environment variables, and you or an administrator can edit its execution role.
- On **AWS Lambda or your own compute**, you must attach the AWS managed policy AWSXrayWriteOnlyAccess to the role your compute uses. For the endpoints, see "Choose an endpoint" in Send telemetry to CloudWatch Omni.

Step 1: Install an instrumentation library

Choose your agent's language in the tabs. The choice applies to every code sample on this page and is remembered on the other pages in this guide. Then choose one of the two libraries; both export the same OTLP traces to CloudWatch and differ in what the spans contain. Zero-code instrumentation covers the frameworks and languages in Supported environments, languages, and frameworks.

Library	What you get	Choose it when
OpenTelemetry distribution (recommended)	The AWS Distro for OpenTelemetry auto-inst ruments model and tool calls with no code changes	You want the least code and standard semantic conventio ns, or your framework and language pair is listed only

Library	What you get	Choose it when
	and records them as spans with `gen_ai.*` attributes. Frameworks with built-in OpenTelemetry tracing, such as Strands Agents, need no extra wiring.	under the distribution in the section called "Supported environments, languages, and frameworks".
OpenInference	Your framework's OpenInference instrumentor runs on top of the distribution. Spans are framework-native: `AGENT`, `LLM`, and `TOOL` span kinds with structured input and output.	You want framework-native spans and can add a package and, for some frameworks, a few lines of startup code.

Important

On startup, the distribution registers the global tracer provider. OpenTelemetry uses only the first provider registered, so do not create another tracer provider or exporter. Attach instrumentors, processors, and custom spans to the distribution's provider, as shown in every snippet on this page.

OpenTelemetry distribution (recommended)

Python

```
pip install "aws-opentelemetry-distro>=0.20.0"
```

Node.js

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation@">=0.13.0"
```

OpenInference

OpenInference is optional. Choose it if you want its framework spans and accept the additional packages and, for some frameworks, startup code. Otherwise, use the OpenTelemetry distribution setup in the previous section.

Install the packages shown, and add any **startup code** to a module that your entry point imports first, before the agent is created. Every install command includes the distribution, so this path does not also need the previous section.

LangGraph

Python

```
pip install "aws-opentelemetry-distro>=0.20.0" openinference-instrumentation-
langchain
```

- No startup code is needed. Invoke the graph after startup, not at import time.

Node.js

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation@">=0.13.0"
 @opentelemetry/api @arizeai/openinference-instrumentation-langchain @langchain/
core@">=1.0.0" @langchain/openai langchain zod
```

agent.ts

```
import { trace } from "@opentelemetry/api";
import { LangChainInstrumentation } from "@arizeai/openinference-instrumentation-
langchain";
import * as CallbackManager from "@langchain/core/callbacks/manager";
import { ChatOpenAI } from "@langchain/openai";
import { createAgent, tool } from "langchain";
import * as z from "zod";

const instrumentation = new LangChainInstrumentation({
  tracerProvider: trace.getTracerProvider(),
});
instrumentation.manuallyInstrument(CallbackManager);

const getWeather = tool(
```

```
  ({ city }) => `The weather in ${city} is sunny.`,
  {
    name: "get_weather",
    description: "Get the weather for a city",
    schema: z.object({ city: z.string() }),
  },
);

const agent = createAgent({
  model: new ChatOpenAI({ model: "gpt-4o-mini" }),
  tools: [getWeather],
});

const result = await agent.invoke({
  messages: [{ role: "user", content: "What is the weather in Seattle?" }],
});
console.log(result.messages.at(-1)?.content);
```

- Requires `@langchain/core` 1.0.0 or later.
- `manuallyInstrument(CallbackManager)` is required for ESM. It attaches OpenInference to the distribution's global tracer provider; do not create another provider or exporter.
- At startup, the distribution attempts to detect another instrumentation library for the same framework and disables its own. If detection fails, add `aws_langchain` to `OTEL_NODE_DISABLED_INSTRUMENTATIONS` so the distribution's instrumentation does not run alongside OpenInference. If the variable is not already set, use `OTEL_NODE_DISABLED_INSTRUMENTATIONS=fs,dns,aws_langchain`.

Strands Agents

Python

```
pip install "aws-opentelemetry-distro>=0.20.0" openinference-instrumentation-
strands-agents
```

tracing.py

```
from opentelemetry import trace
from strands.telemetry import StrandsTelemetry
from openinference.instrumentation.strands_agents import
 StrandsAgentsToOpenInferenceProcessor
```

```
current = trace.get_tracer_provider()
StrandsTelemetry(tracer_provider=current)
current.add_span_processor(StrandsAgentsToOpenInferenceProcessor())
```

- Pass the distribution's provider into `StrandsTelemetry`. A provider that `StrandsTelemetry` creates itself is orphaned and never exports.
- Use this path or the plain distribution, not both. Strands' native tracer occupies the instrumentation point, so the OpenInference processor is inert when native tracing is already active.

Node.js

Use the OpenTelemetry distribution; Strands' built-in OpenTelemetry tracing needs no extra wiring.

CrewAI

CrewAI is Python only.

```
pip install "aws-opentelemetry-distro>=0.20.0" openinference-instrumentation-crewai
 "crewai[bedrock]>=1.10.1" "setuptools<81"
```

- No startup code is needed.
- Set `CREWAI_DISABLE_TELEMETRY=true` where the agent runs.
- `crewai` earlier than 1.10.1 emits no spans.

OpenAI Agents

Python

```
pip install "aws-opentelemetry-distro>=0.20.0" openinference-instrumentation-openai-
agents
```

- No startup code is needed.
- Remove any `set_tracing_disabled(True)` call from your code. While the built-in tracer is disabled, the instrumentor receives no spans.

Node.js

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation@">=0.13.0"
 @arizeai/openinference-instrumentation-openai-agents
```

agent.ts

```
import * as agents from "@openai/agents";
import { trace } from "@opentelemetry/api";
import { OpenAIAgentsInstrumentation } from "@arizeai/openinference-instrumentation-
openai-agents";

const instrumentation = new OpenAIAgentsInstrumentation({
  tracerProvider: trace.getTracerProvider(),
});
instrumentation.manuallyInstrument(agents);

const agent = new agents.Agent({
  name: "Assistant",
  instructions: "You are a helpful assistant.",
});

const result = await agents.run(agent, "What is the capital of France?");
console.log(result.finalOutput);
```

- At startup, the distribution attempts to detect another instrumentation library for the same framework and disables its own. If detection fails, add `aws_openai_agents` to `OTEL_NODE_DISABLED_INSTRUMENTATIONS` so the distribution's instrumentation does not run alongside OpenInference. If the variable is not already set, use `OTEL_NODE_DISABLED_INSTRUMENTATIONS=fs,dns,aws_openai_agents`.

Vercel AI SDK

The Vercel AI SDK is Node.js only. Use the OpenTelemetry distribution:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation@">=0.13.0" ai @ai-
sdk/amazon-bedrock
```

Step 2: Run your agent and deliver its traces

Expand the section for where your agent runs. Each one ends at Step 3.

Start the agent with instrumentation

The distribution must load before your application. AgentCore starter toolkit images configure startup automatically. Use these instructions for custom AgentCore images and your own compute.

Python

```
AGENT_OBSERVABILITY_ENABLED=true \
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_RESOURCE_ATTRIBUTES="service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>" \
OTEL_EXPORTER_OTLP_TRACES_HEADERS="x-aws-log-group=<your-custom-log-group>,x-aws-
log-stream=<your-custom-log-stream>" \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_EXPORTER=otlp \
OTEL_LOGS_EXPORTER=none \
OTEL_METRICS_EXPORTER=none \
opentelemetry-instrument python /path/to/your/agent.py
```

Important

You must set `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true` to keep GenAI content, including model inputs, outputs, and tool calls, on spans. If it is `false` or unset, this content is removed from spans and emitted as OpenTelemetry log events instead.

Node.js

Determine the module format from the compiled application, not the TypeScript source.

For CommonJS:

```
AGENT_OBSERVABILITY_ENABLED=true \
OTEL_RESOURCE_ATTRIBUTES="service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>" \
OTEL_EXPORTER_OTLP_TRACES_HEADERS="x-aws-log-group=<your-custom-log-group>,x-aws-
log-stream=<your-custom-log-stream>" \
```

```
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_EXPORTER=otlp \
OTEL_LOGS_EXPORTER=none \
OTEL_METRICS_EXPORTER=none \
NODE_OPTIONS="--require=@aws/aws-distro-opentelemetry-node-autoinstrumentation/
register" \
node /path/to/your/agent.js
```

For ESM:

```
AGENT_OBSERVABILITY_ENABLED=true \
OTEL_RESOURCE_ATTRIBUTES="service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>" \
OTEL_EXPORTER_OTLP_TRACES_HEADERS="x-aws-log-group=<your-custom-log-group>,x-aws-
log-stream=<your-custom-log-stream>" \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_EXPORTER=otlp \
OTEL_LOGS_EXPORTER=none \
OTEL_METRICS_EXPORTER=none \
NODE_OPTIONS="--experimental-loader=@opentelemetry/instrumentation/hook.mjs --
import=@aws/aws-distro-opentelemetry-node-autoinstrumentation/register" \
node /path/to/your/agent.js
```

Replace `/path/to/your/agent.js` with your compiled entry point. For more ESM startup options, see Using the zero-code option with `@opentelemetry/auto-instrumentations-node`.

Sampling

Sampling determines which traces are recorded and exported.

For the complete list of supported sampler configurations, see https://opentelemetry.io/docs/languages/sdk-configuration/general/#otel_traces_sampler.

We recommend leaving `OTEL_TRACES_SAMPLER` unset. By default, the distribution traces every request that starts at your agent. When a request comes from another traced service, your agent follows that service's decision. Tracing every request keeps metrics that are calculated from spans, such as token usage, accurate.

You can record fewer traces to match your traffic and telemetry requirements, at the cost of incomplete or inaccurate agent metrics. Set the sampler on the service where requests start. For example, if requests start at your hosted agent, you can configure ratio-based sampling:

```
export OTEL_TRACES_SAMPLER=parentbased_traceidratio
export OTEL_TRACES_SAMPLER_ARG=0.9
```

Approximately 90 percent of incoming requests then generate and export traces.

AgentCore

On Amazon Bedrock AgentCore, the runtime exports your agent's spans to CloudWatch; there is no collector to run. Two people are usually involved: the developer sets environment variables on the runtime with every deployment, and an administrator enables trace delivery and grants the runtime role permission once for the account and Region.

Configure the runtime (developer, every deployment)

Set these environment variables on the AgentCore runtime:

```
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
```

- `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true` keeps prompt and response text in the spans, which **Agent traces** and evaluators read. For production redaction and who can read kept content, see the section called “Protect sensitive data”.
- If you build your own container image, its command must start the agent under the OpenTelemetry launcher, as shown in the Amazon EC2, ECS, and EKS sections of Step 2. Images built by the AgentCore starter toolkit do this already.

With the AgentCore starter toolkit:

```
agentcore deploy \
  --env AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
```

- `--env` is repeatable, and its values are not persisted, so pass them on every deployment.
- The toolkit sets `AGENT_OBSERVABILITY_ENABLED` for you; do not pass it with `--env`.

With infrastructure as code, this is the environment block for the runtime resource. AgentCore Runtime injects `AGENT_OBSERVABILITY_ENABLED=true`. When the AWS Distro for OpenTelemetry starts, that flag configures the trace-export defaults, including `OTEL_TRACES_EXPORTER=otlp`, `OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf`, and the regional X-Ray OTLP endpoint. Do not duplicate those managed defaults in the runtime resource; the only variable to set here is the content-extraction opt-out. The name and value are identical across tooling; only the syntax changes.

CDK TypeScript

```
environmentVariables: {
  AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT: "true",
}
```

CDK Python

```
environment_variables={
    "AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT": "true",
}
```

Terraform

```
environment_variables = {
  AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT = "true"
}
```

CloudFormation

```
EnvironmentVariables:
  AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT: "true"
```

Enable trace delivery and grant permissions (administrator, once per account and Region)

AgentCore delivers spans through the account's trace destination, which is the setting Transaction Search turns on. If Transaction Search is not enabled, set the destination:

```
aws xray update-trace-segment-destination --destination CloudWatchLogs --region
 <region>
```

- The change takes effect after about 10 minutes.

> **Note**
>
> You must configure the AgentCore Runtime execution role with the required telemetry permissions. See Execution role for running an agent in AgentCore Runtime.

- A runtime deployed from a container image also needs permission to pull the image from Amazon ECR.

Continue to Step 3.

AWS Lambda

We recommend the latest OpenTelemetry distribution Lambda layer. For the latest ARN for your language and AWS Region, see Lambda layer ARNs. To use a specific version, select the Lambda layer ARN listed in the Python release notes or JavaScript release notes on the GitHub website. Follow the layer instructions to attach it and configure its execution wrapper. Then set the following environment variables.

Python

```
AGENT_OBSERVABILITY_ENABLED=true
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
OTEL_PYTHON_DISTRO=aws_distro
OTEL_PYTHON_CONFIGURATOR=aws_configurator
OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=none
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

The Python Lambda layer enables a limited set of instrumentations by default to reduce cold-start duration. `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=none` enables the agent framework instrumentations, but can increase cold-start duration.

Node.js

```
AGENT_OBSERVABILITY_ENABLED=true
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

If you chose the OpenTelemetry distribution, set `OTEL_NODE_ENABLED_INSTRUMENTATIONS` for your framework. The layer enables only `aws-sdk`, `aws-lambda`, and `http` by default, and it appends `aws-lambda` and `http` to any value you set, so do not include them:

Framework	OTEL_NODE_ENABLED_INSTRUMENTATIONS
LangGraph or LangChain	`aws-sdk,undici,aws_langchain`
OpenAI Agents	`aws-sdk,undici,aws_openai_agents`
Vercel AI SDK	`aws-sdk,undici,aws_vercel_ai`
Strands Agents	Leave unset. Strands Agents emits spans through the tracer provider the layer starts.

With OpenInference, leave `OTEL_NODE_ENABLED_INSTRUMENTATIONS` unset. Install and initialize the OpenInference instrumentor before creating your agent, as shown in Step 1.

Note

You must attach the AWS managed policy AWSXrayWriteOnlyAccess to the function's execution role so that the exporter can send traces to X-Ray.

> **Important**
>
> **Custom trace destination requires a resource policy.** If you set `OTEL_EXPORTER_OTLP_TRACES_HEADERS` to deliver spans to your own log group, you must also add an Amazon CloudWatch Logs resource policy. The policy must allow X-Ray (`xray.amazonaws.com`) to call `logs:PutLogEvents` on that log group. Use the same policy shown in Enabling CloudWatch Transaction Search, with your log group's ARN in `Resource`. Without this policy, X-Ray cannot deliver spans to your log group.

Continue to Step 3.

Amazon EC2

On Amazon EC2, the instance role signs the traces.

Start the agent with its environment

Put the variables in the agent's systemd unit (`Environment=` lines) or start script, and make the launcher the unit's command.

Python

```
AGENT_OBSERVABILITY_ENABLED=true
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
OTEL_PYTHON_DISTRO=aws_distro
OTEL_PYTHON_CONFIGURATOR=aws_configurator
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

Node.js

```
AGENT_OBSERVABILITY_ENABLED=true
```

```
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

- Node.js captures GenAI message content (model payloads and tool responses) by default; `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT` is not required.

Note

You must attach the AWS managed policy AWSXrayWriteOnlyAccess to the instance role so that the exporter can send traces to X-Ray.

Important

Custom trace destination requires a resource policy. If you set `OTEL_EXPORTER_OTLP_TRACES_HEADERS` to deliver spans to your own log group, you must also add an Amazon CloudWatch Logs resource policy. The policy must allow X-Ray (`xray.amazonaws.com`) to call `logs:PutLogEvents` on that log group. Use the same policy shown in Enabling CloudWatch Transaction Search, with your log group's ARN in `Resource`. Without this policy, X-Ray cannot deliver spans to your log group.

Continue to Step 3.

Amazon ECS

On Amazon ECS, the task role signs the traces.

Start the agent with its environment

Put the variables in the `environment` block of the agent's container definition, and make the launcher the container's command.

Python

```
AGENT_OBSERVABILITY_ENABLED=true
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
OTEL_PYTHON_DISTRO=aws_distro
OTEL_PYTHON_CONFIGURATOR=aws_configurator
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

Node.js

```
AGENT_OBSERVABILITY_ENABLED=true
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

- Node.js captures GenAI message content (model payloads and tool responses) by default; `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT` is not required.

Note

You must attach the AWS managed policy AWSXrayWriteOnlyAccess to the task role so that the exporter can send traces to X-Ray.

Important

Custom trace destination requires a resource policy. If you set `OTEL_EXPORTER_OTLP_TRACES_HEADERS` to deliver spans to your own log group, you must also add an Amazon CloudWatch Logs resource policy. The policy must allow X-Ray (`xray.amazonaws.com`) to call `logs:PutLogEvents` on that log group. Use the same policy shown in Enabling CloudWatch Transaction Search, with your log group's ARN in `Resource`. Without this policy, X-Ray cannot deliver spans to your log group.

Continue to Step 3.

Amazon EKS

On Amazon EKS, the pod's service account role signs the traces.

Start the agent with its environment

Put the variables in the `env` block of the agent's container in the pod spec, and make the launcher the container's command.

Python

```
AGENT_OBSERVABILITY_ENABLED=true
AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true
OTEL_PYTHON_DISTRO=aws_distro
OTEL_PYTHON_CONFIGURATOR=aws_configurator
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

Node.js

```
AGENT_OBSERVABILITY_ENABLED=true
```

```
OTEL_RESOURCE_ATTRIBUTES=service.name=<agent-
name>,deployment.environment.name=<stage>,aws.log.group.names=<your-custom-log-
group>
OTEL_EXPORTER_OTLP_TRACES_HEADERS=x-aws-log-group=<your-custom-log-group>,x-aws-log-
stream=<your-custom-log-stream>
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
OTEL_TRACES_EXPORTER=otlp
OTEL_LOGS_EXPORTER=none
OTEL_METRICS_EXPORTER=none
```

- Set the log group and log stream in `OTEL_EXPORTER_OTLP_TRACES_HEADERS` to the destination you want the spans written to.
- Node.js captures GenAI message content (model payloads and tool responses) by default; `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT` is not required.
- The pod's service account must have a role through Pod Identity or IRSA; without one the SDK falls back to the node role, which usually lacks the permission below.

Note

You must attach the AWS managed policy AWSXrayWriteOnlyAccess to the pod's service account role so that the exporter can send traces to X-Ray.

Important

Custom trace destination requires a resource policy. If you set `OTEL_EXPORTER_OTLP_TRACES_HEADERS` to deliver spans to your own log group, you must also add an Amazon CloudWatch Logs resource policy. The policy must allow X-Ray (`xray.amazonaws.com`) to call `logs:PutLogEvents` on that log group. Use the same policy shown in Enabling CloudWatch Transaction Search, with your log group's ARN in `Resource`. Without this policy, X-Ray cannot deliver spans to your log group.

Continue to Step 3.

Step 3: Verify

Use the console's automated verification. In your space, open **Settings › Ingestion** and choose your source: the **Add source** flow's verification step invokes the check for you. It confirms that traces are arriving and that they carry the agent, model, and tool spans CloudWatch Omni reads, and it identifies the failing setting when they do not. Run it after every deployment change.

If verification fails, see the "Troubleshoot missing traces" section of this page.

Verify manually

Traces arrive. Invoke your agent, then open your space and choose **Agent traces**. A trace for the invocation appears within about five minutes. On AgentCore, if the list is empty, check the log group `/aws/bedrock-agentcore/runtimes/<agent-id>-<endpoint-name>` (`<agent-id>-DEFAULT` for the default endpoint), where the runtime writes a `spans` stream; on your own compute, check the log group you set in `OTEL_EXPORTER_OTLP_TRACES_HEADERS`, or the shared `aws/spans` log group if that variable is not set. Transaction Search indexes 1 percent of spans by default for the trace list, so not every invocation is listed until you raise the indexing percentage; the spans themselves are all stored.

The trace is complete. Open the trace. A correctly instrumented agent shows an agent span with non-empty input and output and a child span for each model call and each tool call. With OpenInference, the spans carry `AGENT`, `LLM`, and `TOOL` kinds; with the OpenTelemetry distribution alone, the model calls carry `gen_ai.*` attributes. On AgentCore, the runtime's invoke span is the root and your agent's spans sit beneath it. If prompt and response text is blank, see the section called “Protect sensitive data”.

> **Note**
>
> A successful invocation does not mean that traces arrived. Check for the agent, model, and tool spans, not only for an HTTP 200.

After verification passes, the agent appears in **Agents** in your space, and you can continue to the section called “Analyze agent behavior”.

Troubleshoot missing traces

The following table lists common troubleshooting steps for missing or incomplete traces.

What you see	What to check
No spans are created	**Auto-instrumentation did not start.** Verify that the distribution and required instrumentation packages are installed in the application's runtime environment and that the application uses the documented startup command. Instrumentation must initialize before the agent framework is imported. For Python, also check package-manager environments, development reloaders, and pre-fork workers. For Node.js, verify the CommonJS or ESM preload configuration and confirm that the instrumentation is not disabled. Do not create another tracer provider or exporter. Temporarily use `OTEL_TRACES_EXPORTER=console,otlp` for Python or `OTEL_LOG_LEVEL=debug` for Node.js to inspect startup.
HTTP spans appear, but agent, model, or tool spans do not	Verify that the framework and instrumentation versions are supported and that the corresponding instrumentation package is installed and enabled. Confirm that it is not listed in `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS` or `OTEL_NODE_DISABLED_INSTRUMENTATIONS`. OpenInference startup code must run before the agent is created.
Duplicate agent, model, or tool spans appear	Two equivalent instrumentations may be active. The distribution attempts to detect equivalent instrumentation at startup. If detection fails, add the corresponding AWS instrumentation, such as `aws_langchain`

What you see	What to check
	or `aws_openai_agents` , to `OTEL_NODE_DISABLED_INSTRUMENTATIONS` .
Spans are created locally but do not reach CloudWatch	Verify `OTEL_TRACES_EXPORTER` , `OTEL_EXPORTER_OTLP_PROTOCOL` , the OTLP endpoint, and `OTEL_EXPORTER_OTLP_TRACES_HEADERS` . Check exporter errors and network connectivity. On AWS Lambda, Amazon EC2, Amazon ECS, or Amazon EKS, verify that the compute role has AWSXrayWriteOnlyAccess. On AgentCore, verify the execution-role permissions described in the AgentCore section.
The exporter returns `404`	`OTEL_EXPORTER_OTLP_ENDPOINT` automatically appends `/v1/traces` . If the configured value already includes that path, use `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` instead so the path is not appended twice.
Only the AgentCore runtime invocation span appears	AgentCore telemetry is working, but application auto-instrumentation did not start. Follow the checks in the first row. For a custom image, start the application under `opentelemetry-instrument` or the Node.js register hook. AgentCore enables agent observability, and the distribution derives the OTLP trace-export settings; you do not need to set `OTEL_TRACES_EXPORTER` on the runtime resource.

What you see	What to check
Spans reach the log group but do not appear under **Agent traces**	Verify that Transaction Search is enabled and that the trace destination is CloudWatch Logs. Traces sent before Transaction Search was enabled are not searchable. After changing the destination, wait 10 minutes and invoke the agent again.
Only some requests have traces	Check the caller's sampling decision. Unless the agent is the root service, leave `OTEL_TRACES_SAMPLER` unset so it follows the parent trace.
Prompt, response, or tool content is blank	Set `AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT=true` to keep model payloads and tool request/response data on spans. If content is still missing, verify that the framework instrumentation captures it and that the application does not redact it.

For additional runtime-specific issues, see Troubleshooting Python automatic instrumentation, JavaScript zero-code instrumentation, and JavaScript zero-code configuration.

Supported environments, languages, and frameworks

The environments, languages, and frameworks with a documented path for sending telemetry to CloudWatch Omni.

Environments

Environment	Workload	How telemetry leaves	Signals	Deployment tooling documented	Procedure
Amazon EC2	Applications, AI agents	Applications: CloudWatc	Applications: traces,	AWS CLI, CDK (TypeScri	Applications · AI agents

Environment	Workload	How telemetry leaves	Signals	Deployment tooling documented	Procedure
		h agent on the host. AI agents: direct export from the ADOT SDK.	metrics, logs. AI agents: traces.	pt, Python, Java), Terraform, CloudFormation	
Amazon ECS (Fargate)	Applications, AI agents	Applications: CloudWatch agent sidecar. AI agents: direct export from the ADOT SDK.	Applications: traces, metrics, logs. AI agents: traces.	AWS CLI, CDK (TypeScript, Python, Java), Terraform, CloudFormation	Applications · AI agents
Amazon EKS	Applications, AI agents	Applications: CloudWatch Observability add-on (agent DaemonSet). AI agents: direct export from the ADOT SDK.	Applications: traces, metrics, logs. AI agents: traces.	AWS CLI, Terraform, or CDK for the add-on; kubectl, Helm, or CDK for the application	Applications · AI agents

Environment	Workload	How telemetry leaves	Signals	Deployment tooling documented	Procedure
AWS Lambda	Applications, AI agents	Applications: Lambda delivers segments and logs itself; the ADOT layer adds handler spans. AI agents: the ADOT layer starts the OpenTelemetry runtime and the execution role signs trace exports.	Applications: traces, logs. AI agents: traces.	AWS CLI, CDK (TypeScript, Python, Java), Terraform, CloudFormation, AWS SAM	Applications · AI agents
Amazon Bedrock AgentCore	AI agents	The runtime exports traces.	Traces	AgentCore starter toolkit, CDK (TypeScript, Python), Terraform, CloudFormation	AI agents

Environment	Workload	How telemetry leaves	Signals	Deployment tooling documented	Procedure
Azure VM	Applications	CloudWatch agent on the VM, credentials federated from the VM's managed identity.	Traces, metrics, logs	Azure setup script	Applications
Azure Kubernetes Service	Applications	CloudWatch Observability Helm chart, credentials federated from workload identity.	Traces, metrics, logs	Helm, kubectl	Applications
Google Compute Engine	Applications	CloudWatch agent on the instance, credentials federated from the instance's service account.	Traces, metrics, logs	Google Cloud setup script	Applications

Environment	Workload	How telemetry leaves	Signals	Deployment tooling documented	Procedure
Google Kubernetes Engine	Applications	CloudWatch Observability Helm chart, credentials federated from the cluster's OIDC issuer.	Traces, metrics, logs	Helm, kubectl	Applications

Application languages and frameworks

Language	Auto-instrumentation	Frameworks with documented notes	Procedure
Python	`opentelemetry-instrument` launcher	FastAPI, Flask, Django	Step 4, Python tab
Node.js	`@opentelemetry/auto-instrumentations-node` register hook	Express, gRPC	Step 4, Node.js tab
Java	OpenTelemetry Java agent (`-javaagent`)	Spring Boot, gRPC	Step 4, Java tab
.NET	Automatic instrumentation profiler for ASP.NET Core and gRPC services; SDK	ASP.NET Core, gRPC	Step 4, .NET tab

Language	Auto-instrumentation	Frameworks with documented notes	Procedure
	in code for console applications		
Go	Go has no agent. Initialize the SDK in code and add library instrumentation, such as `otelhttp` or `otelgrpc`.	net/http, gRPC	Step 4, Go tab

- Any of these languages works in any environment in the Environments table. For Python, Node.js, Java, and .NET, the listed frameworks are traced without code changes. For Go, wrap them with the library instrumentation shown in Step 4 of Send application telemetry. Other libraries may be covered by the OpenTelemetry registry for the language.

AI agent languages and frameworks

Each language column lists the libraries documented for that framework: the OpenTelemetry distribution, and the OpenInference instrumentor where one exists. The distribution is required on both paths and is documented for Python and Node.js only: AWS Distro for OpenTelemetry 0.20.0 or later for Python, 0.13.0 or later for Node.js.

Framework	Python	Node.js	Recommended versions	Notes
LangGraph, LangChain	OpenTelemetry · OpenInference	OpenTelemetry · OpenInference	Python: `langchain >=0.3.21, <2` Node.js: `@langchain/core >=1.0.0,< 2.0.0`	Traces through the LangChain instrumentor. Python is zero-code; Node.js needs startup code.

Framework	Python	Node.js	Recommended versions	Notes
Strands Agents	OpenTelemetry · OpenInference	OpenTelemetry	We recommend using the latest available Strands Agents version.	Built-in OpenTelemetry tracing on both languages, so the distribution needs no wiring: the SDK creates the agent, model, and tool spans itself. The Python OpenInference path attaches a span processor to the distribut ion's provider. Use one tracing path, not both, or the processor is inert.
CrewAI	OpenTelemetry · OpenInference	—	`crewai >=1.10.0,<2`	CrewAI is Python only. Zero-code. Requires `setuptool s<81` and `CREWAI_DI SABLE_TEL EMETRY=tr ue .`

Framework	Python	Node.js	Recommended versions	Notes
OpenAI Agents	OpenTelemetry · OpenInference	OpenTelemetry · OpenInference	Python: `openai-agents >=0.3.3,< 1` Node.js: `@openai/agents-core >=0.1.0`	Python is zero-code once `set_tracing_disabled(True)` is removed. On Node.js, registering the instrumentation is not sufficient: the instrumentor attaches through the framework's own tracing processor, so you must also pass the imported SDK namespace to `manuallyInstrument`. Without that call, the agent creates no spans.

Framework	Python	Node.js	Recommended versions	Notes
Vercel AI SDK	—	OpenTelemetry	`ai >=3.3.0,< 7.0.0`	The Vercel AI SDK is Node.js only. The SDK emits its spans automatic ally once the distribution's provider is registered; no per-call telemetry flag is needed. Registering an OpenInference span processor has no effect on this path.

Ask the Omni agent

The Omni agent is the AI assistant built into the CloudWatch Omni web UI. Ask about your systems in plain language, and the agent answers from the telemetry available to it: it writes and runs queries, correlates signals across telemetry, and builds views and dashboards on demand.

> **Note**
>
> CloudWatch Omni supports AI agents on three surfaces. This page covers the agent in the Omni web UI. For AI assistance while developing an AI agent in the IDE extension, see the section called “Develop with the IDE extension”. To use Omni from your own coding agent, see the section called “Use Omni skills”. The Omni agent can also delegate an investigation to the AWS DevOps Agent. See the section called “Investigate with the AWS DevOps Agent”.

Where you can ask

Ask from the input on the Omni home page, or open the chat panel in any thread. Each conversation is saved as a thread you can return to and rename. See the section called “Threads”.

The agent can read what you have open on screen, so you can ask about "this alert" or "the service I'm looking at" without naming it. The agent runs long tasks, such as investigations and dashboard builds, in the background, and you can track them on the **Task Shelf**.

With the Slack integration connected, you can also have the agent respond to your team's Slack conversations. See the section called “Connect Slack”.

What you can ask

The agent can do the same work as the pages in the web UI. Some examples, by task:

To do this	Example prompts
Query your telemetry	"Show me the 5xx errors on checkout-service in the last hour" · "Which services logged timeout errors today?"
Check on something	"How is payment-api doing?" · "Is anything firing right now?"
Investigate a problem	"Why is checkout-service slow?" · "Dig into the errors on orders-api"
Work with Omni alerts	"Alert me when p99 latency on checkout-service stays above 2 seconds" · "Change that alert's threshold to 3 seconds"
Build dashboards	"Build a dashboard for the payment service" · "Save this investigation as a dashboard"
Understand dependencies	"What depends on the orders database?" · "What does checkout-service call?"
Measure AI agent quality	"Score yesterday's traces for correctness" · "Turn these failing traces into a dataset"

To do this	Example prompts
Run commands	"/settings" opens the Settings page. Type "/" to see available commands.

Queries. When the agent writes a query for you, it shows the exact query it ran, so you can check it, refine it, or run it yourself in Explore. For the query syntax, see the section called “Explore and query your telemetry”.

Investigations. A status question gets a quick lookup. When you ask why something is broken, the agent runs a multi-step investigation: it checks for anomalies, maps the services involved, and gathers evidence across your telemetry, then reports the likely cause with the concrete numbers and log lines behind it. It proposes the next step, and you choose the direction. You can also start an investigation from a service page. See the section called “Monitor a service”.

Alerts. The agent creates Omni alerts, not CloudWatch alarms. It can create an alert from a plain-language condition, change an existing one, and answer questions about the alerts you have. Creating and changing are deliberate: the agent grounds the threshold in your data, shows you the specific alert it intends to create or change, and acts only after you confirm. It does not delete alerts. Manage them on the Alerts page. See the section called “Alerts”.

Dashboards. Describe the dashboard you want and the agent composes it in your thread, where you can refine it section by section and save it. It can also turn a finished investigation into a dashboard. See the section called “Dashboards”.

AI agent quality. For the AI agents you monitor, the agent can score traces with your evaluators and add traces to an evaluation dataset — for example, packaging failing runs into a regression set. See the section called “Evaluate agent quality”.

How the agent works with you

Answers are grounded in your live data: the agent re-checks current state rather than answering from earlier conversation, and when a lookup returns nothing, it says so instead of guessing. It also remembers durable context across your threads, such as the services and Regions you work with, so you do not re-explain your environment in every conversation.

The agent is scoped to observability. It declines unrelated work, such as general coding tasks, and offers to help with your telemetry instead.

Limitations

- It reads only your space and your own conversations. It cannot read another person's threads.
- It creates and changes alerts with your confirmation, but never deletes them, and it cannot rename an alert. You delete an alert yourself on the **Alerts** page.
- It can be wrong. It shows its queries and evidence so you can verify. Check both before acting on a finding, and use the feedback controls on a response to rate it.

Permissions and data access

The agent works with your permissions: it can read only the telemetry you have access to (the space's logs and traces, plus metrics queried from CloudWatch Metrics). Its permissions are the same as yours in the Omni web UI. Actions that change something, such as creating an alert, always require your confirmation in the conversation. For permission levels, see the section called "Control access to your space".

Investigations with the AWS DevOps Agent

If your space is connected to the AWS DevOps Agent, the Omni agent automatically delegates root cause analysis to it, and you can track the investigation's progress from the conversation. The investigation runs with the permissions of the DevOps Agent, not yours. It can read CloudWatch Logs log groups directly, and your data scope does not filter its results. See the section called "Investigate with the AWS DevOps Agent".

Investigate with the AWS DevOps Agent

CloudWatch Omni uses AWS DevOps Agent as the native agent for root cause analysis and incident investigation. You must link CloudWatch Omni to a DevOps Agent space before you can start an investigation.

AWS DevOps Agent is an autonomous site reliability engineering (SRE) agent that supports release management and production operations across AWS, multi-cloud, and on-premises environments. For more information, see the DevOps Agent documentation.

Important

Trust boundary and data access

AWS DevOps Agent and CloudWatch Omni operate under different trust boundaries. DevOps Agent investigates with the IAM role on its agent space. You configure this role in the AWS DevOps Agent console. DevOps Agent does not use your space's operator role, the CloudWatch Omni managed policies, or your members' grants. DevOps Agent reads CloudWatch Logs log groups and other sources directly, not through the CloudWatch Dataset. CloudWatch Omni then displays the results.
Data scopes apply only to data read from the CloudWatch Dataset. They do not filter AWS DevOps Agent investigation results or any other data that comes from outside the Dataset.
Any member whose grant includes the AWS DevOps Agent read actions can see every investigation for a connected agent space. This includes investigations that other members started. The read actions are `aidevops:GetBacklogTask`, `aidevops:ListBacklogTasks`, and `aidevops:ListJournalRecords`. The Viewer, Editor, and Space Admin permission levels all include these actions. Connecting an agent space is the access control for this integration. Connect only an agent space whose IAM role reaches data that every member of the space can already see.
Only a Space Admin can connect, change, or remove the DevOps Agent integration. For more information about permission levels, see the section called “Control access to your space”.

Connect an agent space

Deploy DevOps Agent

- A DevOps Agent space must exist in the account you are accessing CloudWatch Omni from. To create one, see Creating an agent space in the AWS DevOps Agent User Guide.

Note

Sign-in must match. If your Omni domain uses IAM Identity Center, configure your DevOps Agent space to use IAM Identity Center as well.

Connect Omni to DevOps Agent

Before you connect, a space administrator must attach the AIDevOpsAgentFullAccess managed policy to your space's operator role. See the section called “Set up Omni”.

1. In CloudWatch Omni, navigate to **Settings** → **Integrations** → **AWS DevOps Agent**. Choose **Select Space**.
2. Choose the Region your agent space is in. Select your agent space. Choose **Connect**.

Start an investigation

- **From the Omni agent.** Describe the symptom in a thread and ask for an investigation. Omni automatically delegates questions related to root cause analysis to DevOps Agent.
- **With the `/investigation` command.** In a thread, enter `/investigation new <description of issue>`.
- **From an alert.** A firing alert offers an **Investigate** action. Investigations are manual: an alert never starts one automatically when it fires.

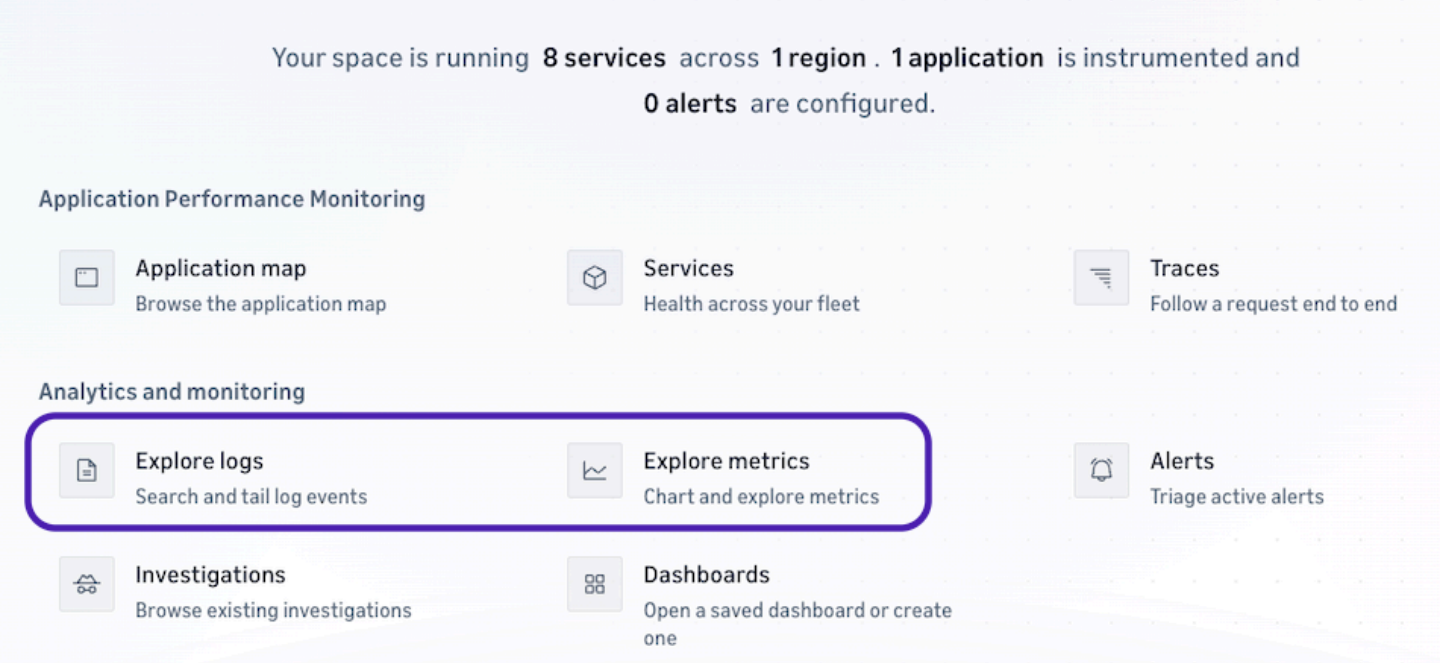
Explore and query your telemetry

Explore is where you query your telemetry directly. For how telemetry reaches your space, see the section called "Send telemetry to CloudWatch Omni". Everything in your space (logs, metrics, and traces) is queryable from one surface, so an investigation can move from a chart into the raw records behind it without changing tools. The other web UI views answer predetermined questions; Explore answers the questions you write yourself.

> **Note**
>
> You query in two languages, split by signal: **CloudWatch Omni SQL** for logs and traces, and **PromQL** (Prometheus Query Language) for metrics. The split holds everywhere you write a query in Omni. An alert condition uses the same two languages (see the section called "Alerts"), and when the Omni agent writes a query for you, it is one you can open and run here (see the section called "Ask the Omni agent").

Run your first query

1. In the navigation, choose **Explore logs** or **Explore metrics**.



1. Choose a telemetry source: **Logs**, **Metrics**, or **Traces**. The navigation entry sets the initial source, and you can switch to any of the three here. The editor opens with a runnable starter query for the source you choose.
2. Write the query in **Builder**, the query editor's guided composer, or switch the **Builder / Code** toggle to enter the query text directly. Builder shows sections for a SQL query, and a tree of metrics, labels, and functions for PromQL.
3. (Optional) Expand the fields on the Browse panel, and select a combination of values to apply as predicates for a targeted query.
4. Read the results below the editor: a table for logs and traces, with a histogram of the result distribution above it, and a line chart for metrics. You can switch the visualization and adjust its options.

You can also describe what you want in natural language and let Omni draft the query for you, then edit and run it like any other. For what the Omni agent can do beyond query drafting, see the section called “Ask the Omni agent”.

Query logs and traces with SQL

A SQL query addresses a signal through its table name: `"logs.default"`, `"traces.default"`. Each result row also carries system fields: `@timestamp` for time filtering, and `@record`, a JSON document holding the fully parsed telemetry record. Quote system field names in backticks:

```
SELECT `@timestamp`, `@record`
FROM "logs.default"
WHERE `@timestamp` BETWEEN now() - INTERVAL '1' DAY AND now()
LIMIT 500
```

By default, the time range you select in Explore applies to your query. You can also bound time explicitly with a `@timestamp` condition, as shown in the preceding example.

Both logs and traces are stored in the same Dataset, so a single query can cover both. Running a query against the `"default"` Dataset looks up a trace by ID across multiple applications and agents.

```
SELECT `@timestamp`, `@record`
FROM "default"
WHERE traceId = '<trace-id>'
LIMIT 100
```

For how application and agent traces relate, see the section called “Monitor a service”.

For the language itself (supported syntax, case sensitivity, bounding time across a join, and unsupported syntax), see the section called “SQL query reference”.

Query metrics with PromQL

For the language and how CloudWatch metrics support it, see Query metrics with PromQL and Running PromQL queries in Query Studio in the Amazon CloudWatch User Guide. **Builder** composes the expression — pick a metric, narrow by label values, and apply functions — or switch to **Code** and write the expression directly. In the Browse panel you can select label values before choosing a metric, and the metric list narrows to the metrics that carry them.

Discover fields

Fields differ across sources. For whichever signal you are querying (logs or traces), the Browse panel aggregates fields from all sources of that type, so you see every field in your current signal in one place. The field list and each field's values narrow to the rows matching your current query, so selecting a value tightens the query and the field list together.

For metrics, the Browse panel lists metrics and their labels instead of fields, and you can select label values to narrow the metric list.

To ask the same question in the query text instead, see the section called “SQL query reference”.

Work with results

In the results table, you can resize, reorder, and hide columns, filter the rows, highlight keywords, and export the results. For metrics, you can adjust the chart's legend and display options, and you can change the visualization type for any result.

CloudWatch Omni SQL limits

- If you do not set `LIMIT`, results are capped at 10,000 rows. Set an explicit `LIMIT` for a known row count, up to a maximum of 100,000.
- Results are paginated: each page returns up to 1,000 rows; page through the rest with the next-page token (`nextToken` on the API).
- A query string can be up to 64,000 characters.
- A query times out after 60 minutes.

Act on what you find

- **Alert on it.** An Omni alert takes a query as its condition, in the same two languages. You can start an alert directly from the query you ran. See the section called “Alerts”.
- **Keep it on a dashboard.** Save the view so the query and its visualization become a shared panel. See the section called “Dashboards”.
- **Hand it to the agent.** Ask the Omni agent to investigate what the query surfaced. See the section called “Ask the Omni agent”.
- **Automate it.** Run the same query from your own tooling through the public API, on a schedule or as part of a workflow.

SQL query reference

CloudWatch Omni supports SQL for querying logs and traces. This page covers CloudWatch Omni SQL: the supported syntax and the behaviors that differ from other SQL dialects. To build SQL queries interactively, use Explore. See the section called “Explore and query your telemetry”.

SQL is the same language wherever you write a query in Omni: in Explore, in an alert condition, and through the public API.

Tables to query

A query addresses a signal through its table in your space's CloudWatch Dataset. The Dataset is named `default`:

FROM table	Reads
`logs.default`	Log records only.
`traces.default`	Traces only — both application spans and AI agent spans.
`default`	Logs and traces in one query. Use this when you do not know which signal holds the field you want.

Prefer a concrete signal type whenever possible. Such queries are faster because they do not scan other signals' data.

System fields

Every row carries system fields alongside the fields your telemetry brought with it: `@timestamp`, `@ingest_time`, and `@telemetry_type` on every signal, and `@message` on log records. The `@entity`, `@data_source_name`, `@data_source_type`, `@data_format`, `@aws.region`, `@aws.account`, `@logGroupId`, `@logGroupName`, `@logStream`, and `@logStreamId` system fields are also supported.

Quote any field name that is not a plain identifier: one with a leading `@`, a dot, or a hyphen. Backticks and double quotes both work:

```
SELECT `@timestamp`, `@message`
FROM "logs.default"
WHERE `@timestamp` BETWEEN NOW() - INTERVAL '1' DAY AND NOW()
LIMIT 500
```

Quoting decides whether Omni reads a token as a column or as text:

- Backticks and double quotes mark an **identifier**. `` ` status ` `` and `"status"` both name the column `status`.

- Single quotes mark a **string literal**. `'ERROR'` is the text ERROR.
- Brackets look up a key inside an attribute map, and the key is always a string literal, so it takes single quotes: `attributes['http.status_code']`.

Because double quotes are an identifier, `WHERE status = "ERROR"` compares the `status` column against a column named ERROR, not against the text ERROR. Write `WHERE status = 'ERROR'` instead.

Field names are case sensitive: `traceId` and `traceid` are not interchangeable.

What a wildcard SELECT returns

Because the Dataset contains records from different sources, each can have a different set of fields. That is why `SELECT *` does not project all fields as columns. It returns two columns representing a full record: `@timestamp` and `@record` — a JSON document holding the fields your telemetry brought with it. System fields are not part of `@record`; select them by name.

```
@timestamp                  | @record
2026-08-26T18:04:11.230Z    | {"request_id":"abc-123","path":"/api/auth/
login","status_code":500, ...}
```

Name the fields you want as columns:

```
SELECT
  request_id,
  path,
  status_code
FROM "logs.default"
WHERE `@timestamp` >= NOW() - INTERVAL '15' MINUTE
```

You can filter on a field whether or not you select it, so `WHERE path = '/api/auth/login'` works under `SELECT *`. Only the projection changes, not what you can reference.

`@record` is an output column only. It cannot feed a function, a comparison, `WHERE`, `GROUP BY`, `ORDER BY`, or a join condition. Name the field you want instead.

`@message` is a system field that holds the raw record. It is useful for unstructured records, where Omni cannot extract any columns and `@record` is empty (`{}`).

Nested attributes

Read into an attribute map with bracket syntax:

```
SELECT
  attributes['http.status_code'],
  resource['attributes']['service.name']
FROM "traces.default"
WHERE `@timestamp` >= NOW() - INTERVAL '1' HOUR
```

Bracket syntax accepts only string literals as the key, so use single quotes for subscript access. Dots within the string literal are allowed when they are part of the key name, for example `'service.name'`.

Column types

No schema is required for a Dataset in Omni. Omni SQL identifies the correct type from the query itself and applies it automatically. For example, when a `WHERE` filter compares against a numeric value such as `WHERE latency > 100`, values are cast to the appropriate type.

A Dataset may contain records from different sources with different fields. The same field name may appear with different types. For example, `status` may hold enumeration values in some records (`OK`, `ERROR`) and numeric values in others (0, 1). In such cases, Omni selects the most appropriate type to execute the query correctly.

In ambiguous cases, use the `CAST` operator.

Time filtering

The time range you select in Explore automatically applies to your query. You can also bound time in the query text with a `@timestamp` condition, using `BETWEEN` or comparison operators, and `NOW()` with `INTERVAL` arithmetic. A `@timestamp` condition in the query takes precedence over the range selected in Explore. Other timestamp functions are also supported.

In a join, bound each table separately. A condition on one side does not constrain the other, so a join with a single bound reads the whole retention window on the unbounded side:

```
SELECT a.request_id, a.path, b.user_id
FROM "logs.default" a
JOIN "logs.default" b ON a.request_id = b.request_id
WHERE a.`@timestamp` >= NOW() - INTERVAL '1' HOUR
```

```
  AND a.`@timestamp` <= NOW()
  AND b.`@timestamp` >= NOW() - INTERVAL '1' HOUR
  AND b.`@timestamp` <= NOW()
```

Discover fields

Wrap any query in `EXPLAIN (ANALYZE_FIELDS)` to get one row per field observed in the rows that query selects, instead of the data rows:

```
EXPLAIN (ANALYZE_FIELDS)
SELECT *
FROM "logs.default"
WHERE `@timestamp` >= NOW() - INTERVAL '1' HOUR
LIMIT 500
```

Your conditions, time bounds, and `LIMIT` all apply, so you can ask which fields appear on one service's error logs in the last hour rather than which fields exist anywhere. the section called "Explore and query your telemetry" covers how the Browse panel runs this for you.

`EXPLAIN (ANALYZE_FIELDS)` is not billed or metered: it samples a bounded slice of your data, so you can run it freely to explore a Dataset's fields.

Supported and unsupported syntax

CloudWatch Omni SQL supports the constructs this page demonstrates — `SELECT` with `WHERE`, `GROUP BY`, `ORDER BY`, and `LIMIT`, joins, and aggregate functions with `FILTER` — as well as window functions, `UNION` and `UNION ALL`, and `CAST`. `EXPLAIN (ANALYZE_FIELDS)` is the only supported form of `EXPLAIN`.

The engine rejects unsupported statements before they run, returning an unsupported-statement error rather than an empty result, so you can distinguish a rejected query from one that matched nothing.

Examples

Slowest agent runs in the last hour. Each span's duration is the difference between its `startTimeUnixNano` and `endTimeUnixNano`, and the agent's name comes from its resource attributes:

```
SELECT
```

```
  resource['attributes']['service.name'] AS agent,
  `@timestamp`,
  (endTimeUnixNano - startTimeUnixNano) AS duration_ns
FROM traces.default
WHERE `@timestamp` >= NOW() - INTERVAL '1' HOUR
ORDER BY duration_ns DESC
LIMIT 20
```

Error rate by service. Counting over a grouped select:

```
SELECT resource['attributes']['service.name'] AS service,
       COUNT(*) AS spans,
       COUNT(*) FILTER (WHERE status['code'] IN ('2', 'ERROR')) AS errors
FROM traces.default
WHERE `@timestamp` >= NOW() - INTERVAL '1' HOUR
GROUP BY resource['attributes']['service.name']
ORDER BY errors DESC
```

One request across an application and an agent. Both carry the same trace ID when context propagates:

```
SELECT *
FROM "traces.default"
WHERE
  `@timestamp` >= NOW() - INTERVAL '1' HOUR
  AND traceId = '<trace-id>'
```

Next steps

- To build a query interactively, see the section called “Explore and query your telemetry”.
- To alert on a query result, see the section called “Alerts”.
- To put a query on a dashboard, see the section called “Dashboards”.

Dashboards

A dashboard is a saved arrangement of panels (charts, tables, maps, and text) that shows your team one agreed view of a system. Most panels are backed by a query against your space's telemetry. Build one per audience: a service's health for its on-call, an agent fleet's quality trends for its owners, a cross-signal view for an operations review.

> **Note**
>
> Omni dashboards are separate from the dashboards in the CloudWatch console. Your existing CloudWatch dashboards keep working unchanged (see *CloudWatch Omni*).

Every dashboard is shared

Every dashboard belongs to your space, and every member of the space can see it. Name dashboards so others can find them.

Pages and dashboards

You build and edit on a **page**, the live working surface on a canvas with a shared time range where you arrange related panels. Saving a page makes it a dashboard. Opening a saved dashboard gives you a working copy on a fresh page: you can rearrange and experiment without changing what the rest of the team sees. When you save, you choose whether to update the original — this is how you edit a shared dashboard — or keep your version as a new dashboard. Omni saves your editing progress as you work, so if you are signed out you can resume in your most recent thread.

What goes on a dashboard

Any view you work with in the Omni web UI is a panel, and application and agent telemetry mix on the same dashboard: a service's request charts next to its agent's quality trends. Panels include query visualizations from Explore, the application map and service views, trace and session views, alert status, agent fleet and evaluation views, and text panels for notes. For the query languages behind query-backed panels, see the section called “Explore and query your telemetry”.

Create a dashboard

You can create a dashboard in three ways, depending on where you start:

- **Save the view you are in.** From any page, choose **Save** in the toolbar. An Explore query or an investigation's working view becomes a shared dashboard.
- **Start in the Dashboards page.** Choose **Create dashboard**, then start from an empty canvas or a starter template, and add panels.

- **Describe it.** Ask the Omni agent to build a dashboard from a plain-language description, or to turn a finished investigation into one, then refine and save the result. See the section called “Ask the Omni agent”.

Manage your dashboards

The **Dashboards** page lists every dashboard in the space. From it you can open or delete a dashboard, and search the list.

> ⚠ **Deletion is permanent**
>
> Deleting a dashboard is permanent and removes it for everyone in the space.

Threads

A thread is your workspace in the CloudWatch Omni web UI: a conversation with the Omni agent together with the canvases, tabs, and views you open while you work. Omni saves the whole thread as you work — not only the chat — so you can close the browser mid-investigation and resume where you left off, on any device where you sign in to your space. Keep one thread per problem.

What a thread holds

- **The conversation.** Your exchange with the Omni agent, including the queries it ran and the evidence behind its answers.
- **Canvases and tabs.** The views you open while investigating (charts, log tables, maps, trace views), arranged on canvases, across as many tabs as you need.
- **History.** Every canvas you visit in a thread is kept in order, so you can step back through the investigation.
- **Investigations.** An investigation starts in a thread, and that thread is marked in your threads list so you can find the investigation again later. An investigation is not locked to that thread. You can open it from another thread, and the Investigations list can show investigations from all your Omni threads.

Start a thread

Choose **New thread** in the sidebar or on the **Threads** page. Asking from the input on the Omni home page also starts a thread.

You can also start from a finding instead of a blank page: choose **Investigate** on an alert, or select an anomalous range on a chart and choose **Investigate this signal**. The investigation runs in a thread, with the finding as its starting context. For what the agent can do in the conversation and how an investigation works, see the section called “Ask the Omni agent”.

Manage your threads

The **Threads** page lists your threads, and the sidebar shows the recent ones. Omni names a thread from the conversation; rename it to something your team would search for. From the list you can:

- **Resume** a thread to continue working in it.
- **Rename** or **Delete** a thread. Deleting cannot be undone, and you always keep at least one thread.
- **Search** your threads by name.

Collaborate and share

To share work from a thread, use a dashboard or a link:

- **Save a canvas as a dashboard.** The durable way to share an investigation's view: every member of the space can open a dashboard. See the section called “Dashboards”.
- **Copy a canvas or panel link.** From a canvas or panel menu, copy its URL. The link carries the view, not the conversation. Opening it adds that view to the current thread.

Alerts

An alert monitors your telemetry and notifies you when it crosses a threshold you set. You write a query to select what to monitor, set a condition that compares the query result against a threshold, and choose how often CloudWatch Omni evaluates it. When the condition is met, CloudWatch Omni sends a notification to the destinations you specify. Alerts read the same telemetry as the rest of CloudWatch Omni, so one alerting system covers both your applications and your AI agents: error rates and latency alongside evaluation scores and token usage.

> **Note**
>
> Omni alerts are separate from CloudWatch alarms. Your existing CloudWatch alarms keep working unchanged, and you manage them in the CloudWatch console. See *CloudWatch Omni*.

Alert query

An alert monitors the results of a query that you provide. You write the query in the same languages you use in CloudWatch Omni: SQL for logs and traces, and PromQL for metrics. You can alert on anything you can query in Explore. Start from a query you have already run there.

Thresholds and alert states

Each evaluation compares the query result against a threshold. You set a warning threshold, a critical threshold, or both. A single alert carries both tiers: a result that crosses the warning threshold puts the alert in the `WARNING` state, and a result that crosses the critical threshold escalates it to `CRITICAL`. At any moment an alert is in one of four states: `OK`, `WARNING`, `CRITICAL`, or `NODATA` (the last when the query returned nothing to evaluate). You choose how missing data is treated (as breaching at a state you set, or not), so a pipeline that stops sending data can itself raise the alert.

A threshold applies to either the values a query returns or the number of rows it returns. You choose which comparison the alert makes when you set the threshold.

- **Field value** — the alert reads a column that you name from each row the query returns and compares that value against the threshold.
- **Row count** — the alert compares the number of rows the query returned and reads no column. Use this to alert on how many items match a condition.

Evaluation timing

Three settings control when the alert changes state:

- **Evaluation interval** — how often the query runs.
- **Pending duration** — how long results must stay past the threshold before the alert fires, so a single spike does not notify anyone.

- **Recovery duration** — how long results must stay clear before the alert returns to OK, so a flapping signal does not resolve and re-fire repeatedly.

One alert, many series

When the query returns one result per service, agent, or other grouping, the alert tracks each series as a **contributor** with its own state. The alert's state reflects its worst contributors, and its detail view shows which series are breaching. One alert monitors a whole fleet without a copy per service.

Create an alert

You can create an alert in three ways, depending on where you start:

- **From Explore.** Run the query, then create the alert directly from it: the query you ran becomes the alert's query. See the section called “Explore and query your telemetry”.
- **From the Alerts page.** Choose **Create alert**. A query editor opens in place: write and run the query, and the alert configuration form appears with a preview of your thresholds against recent data.
- **Ask the Omni agent.** Describe what you want to be alerted on in plain language. The agent bases the threshold on your data, shows you the alert it intends to create, and creates it only after you confirm. See the section called “Ask the Omni agent”.

Then name the alert, set the thresholds and evaluation timing, and add destinations. Choose the name deliberately: an alert cannot be renamed later.

Every alert also carries an **access profile** (the identity it evaluates its query and sends notifications as), and Omni selects one for you by default. You can select a different profile for finer-grained control. If an alert reports no data from the moment you create it, rather than going quiet after working, check its profile before the query. See the section called “Access profiles”.

Creating an alert requires permission to create alerts in the space and access to a profile that carries the grants below. Whichever profile the alert uses, it must carry these grants:

- Grants that allow the alert and you, its creator, to assume the profile.
- A grant that allows the query actions, `cloudwatch:StartTelemetryQuery` and `cloudwatch:GetTelemetryQueryResults`.

- If the alert notifies a Slack channel, a grant that allows the Slack integration, scoped to that channel. See the section called “Connect Slack”.

For how to author grants, see the section called “Custom grant actions”.

Send notifications

Notification rules route by state: send warning notifications to one destination and critical notifications to another, and notify on recovery to OK so the person who was notified knows the alert cleared. A destination is a Slack channel (connected through the Slack integration, see the section called “Connect Slack”) or an Amazon SNS topic.

Delivery to a Slack channel also requires that the Region's Amazon CloudWatch bot has been invited to that channel. For the invite command and the bot names by Region, see the section called “Connect Slack”.

A new alert sends nothing until you add a notification rule, and you can disable an alert's notifications without deleting it. The alert keeps evaluating and showing state.

Manage your alerts

The **Alerts** page lists every alert in the space with its state and recent state changes. From the list you can open an alert's detail view (the query result charted with the thresholds drawn on it, the alert's state timeline, and its contributors), edit the alert, disable or re-enable its notifications, tag it, or delete it. Deleting cannot be undone.

Every state change, configuration change, and notification an alert sends is recorded as telemetry in your space and kept for 90 days. Query it with the same SQL you use for the rest of your telemetry (how often an alert fired last quarter, which contributors breach most often, whether a notification was delivered), and join it against your application telemetry in a single query. See the section called “Explore and query your telemetry”.

An alert evaluates against your space's CloudWatch Dataset, so it can fire on telemetry from a centralized source account. See the section called “Set up Omni for your organization”.

Investigate a firing alert

Investigations are manual: an alert never starts one automatically when it fires. When your space is connected to the AWS DevOps Agent, you start the investigation yourself from a firing alert's

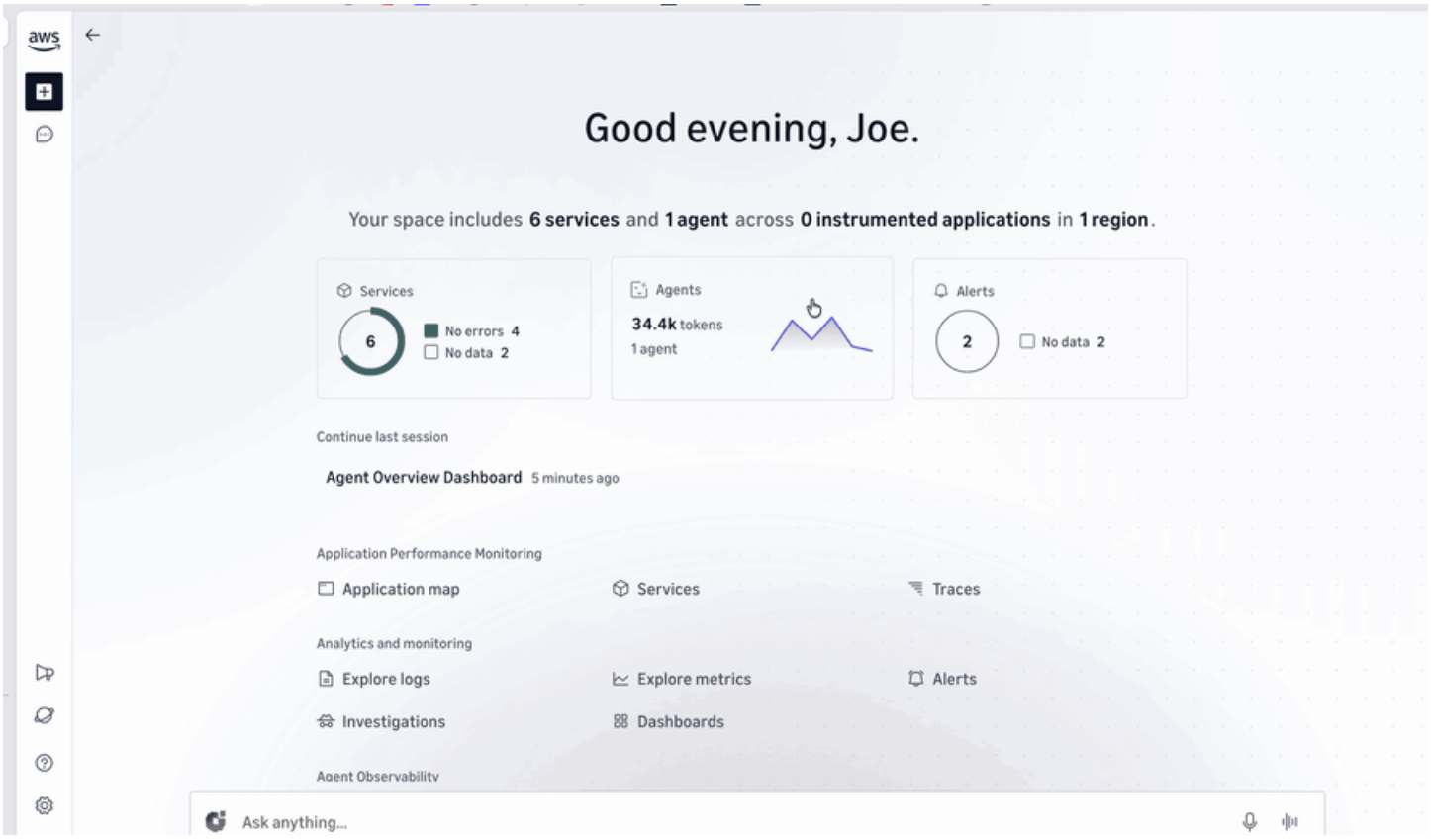
Investigate action. The investigation is scoped to that alert and seeded with the alert's context: its state and what fired. See the section called “Investigate with the AWS DevOps Agent”.

Monitor AI agents

CloudWatch Omni agent observability includes tracing, evaluations, datasets, and experiments. Agent telemetry is ingested into the same CloudWatch Omni space and CloudWatch Dataset as your other telemetry.

> **Note**
>
> **New to agent monitoring?** Follow the section called “Tutorial: Fix a production quality regression”. It starts with installing the IDE extension, then takes you through your first traces and your first evaluation.

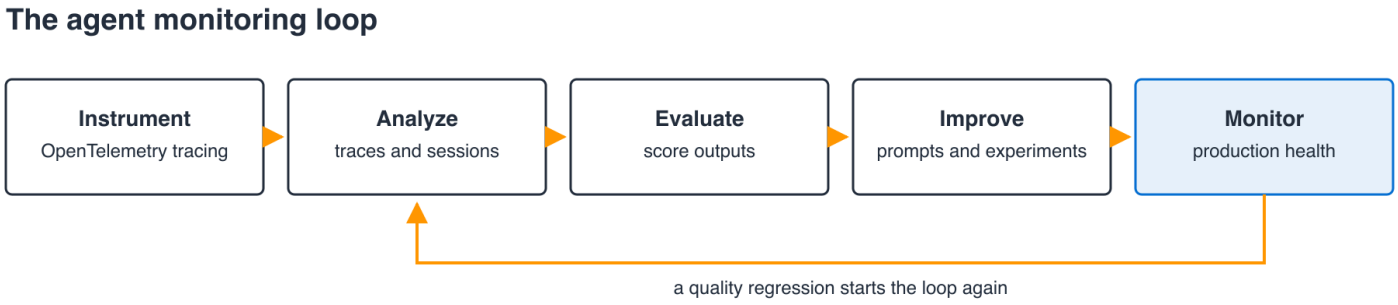
The following animation shows the agent monitoring workflow in the Omni web UI: agent metrics and latency, evaluation scores across evaluators, per-trace evaluator detail, evaluation datasets, and a prompt playground comparison that scores two model configurations.



What you can do

- **Instrument your agent** — add OpenTelemetry tracing so agent runs arrive as traces. See the section called “Send AI agent telemetry”.
- **Analyze agent behavior** — explore traces, sessions, and the agent topology to debug failures and latency. See the section called “Analyze agent behavior”.
- **Evaluate agent quality** — score outputs with built-in or custom evaluators. These include open-source evaluators from DeepEval and AutoEval (see the section called “Evaluators and evaluations”). You can also build datasets from real runs and compare changes with experiments. See the section called “Evaluate agent quality”.
- **Develop in the IDE** — test your agent locally, manage prompts, and get AI assistance during development. See the section called “Develop with the IDE extension”.
- **Monitor in production** — track aggregate agent health. See the section called “Monitor agents in the web UI”.

The following diagram shows how these stages form a loop:



How an agent run is represented

Omni uses OpenTelemetry concepts to represent your agent's activity. OpenTelemetry is an open standard for telemetry collection. These definitions apply everywhere you work with traces in Omni, for both agents and applications.

- **Trace** — the complete record of one unit of work: a request that moves through your services, or a single agent run.
- **Span** — one step inside a trace, such as a service call, a model invocation, a tool call, or a retrieval. A span records timing, inputs and outputs, and attributes. Spans nest to show which step calls which other step.

- **Session** — a sequence of related traces that belong to the same conversation or user interaction. A session lets you follow a multi-turn exchange across several runs.

You can explore these concepts in more detail:

- Traces and spans in the section called “Agent traces”.
- Sessions, and how Omni groups traces into them, in the section called “Agent sessions”.
- How your agent's components call each other in the section called “Agent topology”.

Where to start

- **New to agent monitoring** — follow the section called “Tutorial: Fix a production quality regression”. The tutorial starts with how to install the extension. Then it takes you through your first traces and evaluations.
- **Agent already sending traces** — go directly to the section called “Analyze agent behavior” to debug runs, and to the section called “Evaluate agent quality” to measure quality.
- **Operating agents in production** — start from the section called “Monitor agents in the web UI”, and use the section called “Tutorial: Fix a production quality regression” when you find a quality regression.

Available across two surfaces

Agent monitoring is available through two surfaces that serve different workflows:

- **Omni web UI** — a production-time experience for monitoring agents, with aggregate and fleet-level views alongside trace and evaluation analysis.
- **IDE extension** — a development-time experience for the local build-test-iterate loop, including instrumenting your agent, testing it locally, managing prompts, and running experiments. The extension works in VS Code, Kiro, and Cursor.

Both surfaces connect to the same space, so the data you collect is available wherever you work. Use the IDE extension for local instrumentation and troubleshooting during development. Use the Omni web UI for production monitoring, fleet-level views, and online evaluations. The following table is the complete per-capability reference.

Feature availability by surface

Capability	Omni web UI	IDE extension
Agent traces	Yes	Yes
Agent sessions	Yes	Yes
Agent topology	Yes	Yes
Datasets	Yes	Yes
Evaluators and evaluations	Yes	Yes
Online evaluations	Yes	No
Prompt playground	Yes	Yes
Experiments	No	Yes
Prompt management and versioning	No	Yes
Test your agent locally	No	Yes
Agent-assisted instrumentation setup	No	Yes
Prompt recommendations	No	Yes
Aggregate and fleet views	Yes	No

Where your agent runs

You can monitor agents wherever they run. Agents hosted on Amazon Bedrock AgentCore send traces and sessions with minimal setup (see the section called “Send AI agent telemetry”). Agents built on other frameworks or runtimes connect through OpenTelemetry instrumentation. You can also observe and evaluate agents deployed on other compute platforms, such as Amazon EC2,

Amazon ECS, Amazon EKS, and AWS Lambda. For every environment, language, and framework this supports, see the section called “Supported environments, languages, and frameworks”.

How Omni relates to AgentCore and generative AI observability

Amazon Bedrock AgentCore is a platform for building, deploying, and operating AI agents, with any framework and any foundation model. Omni works with AgentCore in two ways:

- **As a place your agents run.** Agents hosted on AgentCore Runtime send traces and sessions to CloudWatch with minimal setup, and Omni monitors them the same way it monitors agents in any other environment. See the section called “Send AI agent telemetry”.
- **As the service behind evaluations.** Omni's evaluators are served through AgentCore Evaluations: built-in, third-party, and custom evaluators come from a single catalog, and on-demand and online evaluations run through it. See the section called “Evaluators and evaluations”.

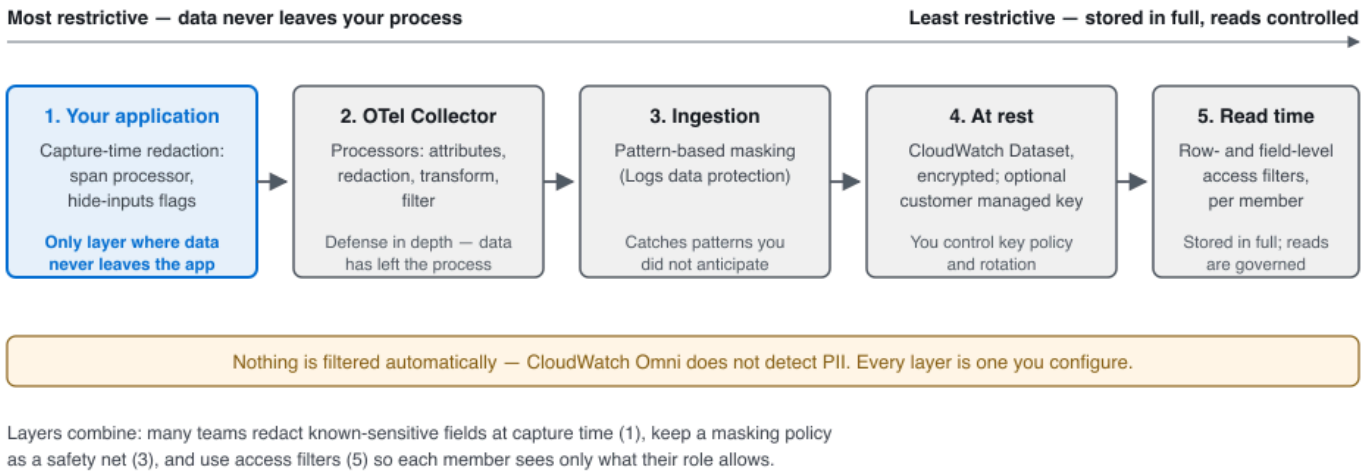
Amazon CloudWatch also provides generative AI observability in the AWS Management Console, with prebuilt views for model invocations and AgentCore agents. It is a separate experience from Omni, and both read the telemetry you ingest into CloudWatch, so an AgentCore agent's traces appear in each. Use generative AI observability for its prebuilt views. Use Omni to work in a space — agent telemetry alongside your application telemetry — and for evaluations, evaluation datasets, experiments, and the IDE extension.

Protect sensitive data

Agent telemetry can contain sensitive information: prompts and responses often include user messages, retrieved documents, and personal data. You control what is captured and who can see it. CloudWatch Omni does not automatically detect or redact personally identifiable information (PII); no data is removed unless you configure it. Choose where in the pipeline filtering happens based on your compliance requirements.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or into free-form text fields such as a **Name** field. This includes when you work with CloudWatch Omni or other AWS services using the Omni web UI, the AWS Management Console, the API, the AWS CLI, or the AWS SDKs. Any data that you enter into tags or free-form text fields used for names might be used for billing or diagnostic logs. CloudWatch Omni accepts tags on several resource types, and free-form names on spaces, grants, access profiles, dashboards, and alerts. For the resource types that accept tags, see the section called “Custom grant actions”.

You can apply controls at five points, from most restrictive to least. Each layer involves a trade-off between eliminating data and keeping it available for debugging.



Layers combine: many teams redact known-sensitive fields at capture time (1), keep a masking policy as a safety net (3), and use access filters (5) so each member sees only what their role allows.

At capture time (most restrictive)

Strip or redact fields before spans leave your application, using standard OpenTelemetry mechanisms. Sensitive data filtered here is redacted in process and never leaves your application, so this is the only layer that satisfies a requirement that the data never be exported.

`AWS_REDACT_SPAN_ATTRIBUTES` takes a comma-separated list of span attributes to redact. Matching values in spans, span events, and span links are all replaced with `REDACTED`. This applies to all span attributes, not only those from ADOT instrumentation.

```
export AWS_REDACT_SPAN_ATTRIBUTES=gen_ai.input.messages,gen_ai.output.messages
```

The variable also accepts wildcard patterns, so you can redact a group of attributes without naming each one. For example, to redact all input and output message attributes from a third-party instrumentation library:

```
export AWS_REDACT_SPAN_ATTRIBUTES='llm.input_messages.*,llm.output_messages.*'
```

Warning Redaction occurs in the agent process, before telemetry is exported. This might affect other integrations that rely on these attribute values.

`AWS_REDACT_SPAN_ATTRIBUTES` applies equally to OpenInference-instrumented agents running on Amazon Bedrock AgentCore.

To redact all prompt and response content from an agent instrumented with OpenInference, set the OpenInference environment flags before your agent starts. Content is replaced with __REDACTED__ before spans are created:

```
OPENINFERENCE_HIDE_INPUTS=True
OPENINFERENCE_HIDE_OUTPUTS=True
```

For finer control, add a custom span processor that removes specific attributes. This is standard OpenTelemetry and works with the AWS Distro for OpenTelemetry (ADOT), OpenInference, and other OpenTelemetry SDKs. The span-processor API differs by language, and the following example is Python. **JavaScript does not provide an equivalent custom span-processor approach, so we recommend the environment variable approach described above.**

```
from opentelemetry.sdk.trace import SpanProcessor, Span, ReadableSpan

class SensitiveDataRedactingProcessor(SpanProcessor):
    """Redacts sensitive attributes from spans before export."""

    REDACT_KEYS = {"gen_ai.input.messages", "gen_ai.output.messages",
                   "user.email", "http.request.header.authorization"}

    def on_start(self, span, parent_context=None):
        pass

    def _on_ending(self, span: Span):
        if not span.attributes:
            return
        redacted = {k: v for k, v in span.attributes.items()
                    if k not in self.REDACT_KEYS}
        if len(redacted) != len(span.attributes):
            span._attributes = redacted  # noqa: SLF001

    def on_end(self, span: ReadableSpan):
        pass

    def shutdown(self):
        pass

    def force_flush(self, timeout_millis=None):
        return True

# Wire up before your agent starts:
```

```
from opentelemetry import trace
provider = trace.get_tracer_provider()
provider.add_span_processor(SensitiveDataRedactingProcessor())
```

This example requires `opentelemetry-sdk` version 1.39.0 or later.

Note Capture-time redaction is irreversible. Redacted data is not available for root-cause analysis later. If a trace shows blank input and output that you expected to see, check these flags first. See the section called “Send AI agent telemetry”.

At the collector (defense in depth)

If you run an OpenTelemetry Collector, its built-in processors provide a second redaction layer before data reaches CloudWatch:

- `attributes` — remove or modify specific attributes.
- `redaction` — drop attributes that are not on an allow list.
- `transform` — replace values using regular expressions.
- `filter` — drop entire spans that match a condition.

At this layer, data has already left your application process. Use collector processors as an additional safety net. They are not sufficient on their own if your compliance model requires that data never leave your infrastructure.

For each processor's configuration, see its documentation: attributes, redaction, transform, and filter.

This layer is available wherever you run the collector yourself, for example on Amazon EC2, Amazon ECS, or Amazon EKS. If your agent runs on Amazon Bedrock AgentCore Runtime, you do not manage a collector there, so redact at capture time instead.

At ingestion: pattern-based masking

For log data in Amazon CloudWatch Logs, CloudWatch Logs data protection can detect and mask sensitive patterns, such as Social Security numbers and credit card numbers, as it is ingested. For more information, see Help protect sensitive log data with masking in the *Amazon CloudWatch Logs User Guide*.

> **Important**
>
> Log data that a CloudWatch Logs data protection policy masks stays masked in your space. It is forwarded to the CloudWatch Dataset in masked form, and Omni cannot unmask it. Unmasked values are available only in CloudWatch Logs, to principals with the `logs:Unmask` permission.

At read time: row-level data scope

Telemetry is stored in full, and a member's grant controls what their queries and views return. A grant's advanced data scope can limit a member to specific signal types (logs, traces) and to rows matching conditions on agent, service, account, or field value. Configure it when you add or edit a member. See the section called “Limit what members can see”.

At rest: encryption

Telemetry is encrypted in transit with TLS and at rest in the CloudWatch Dataset, where you can use a customer managed AWS KMS key. For the encryption model, see the section called “Data encryption in CloudWatch Omni”.

Data sent to the judge model

Evaluators that use a model as a judge send data to that model: the evaluator prompt and the trace content it evaluates go to the judge model you configure, and to the provider that serves it. If your compliance model restricts where trace content can flow, choose the judge model accordingly, or restrict evaluation to traces that do not contain sensitive content. For how evaluators work and which models you can choose, see the section called “Evaluators and evaluations”.

Choose the right layer

Requirement	Layer	Trade-off
Sensitive data must never leave the application	Capture time	Redacted data is unavailable for debugging
Extra safety net on the export path	Collector processors	Data has already left the process

Requirement	Layer	Trade-off
Catch sensitive patterns you did not anticipate	Ingestion-time masking	Data has been sent; masking depends on pattern detection
Store in full; limit which rows a member sees	Per-grant row-level data scope	Data is stored; controls govern reads
Control the encryption keys	Customer managed KMS key	You manage key policy and rotation

Analyze agent behavior

Use these views to understand what your agent did and why: to debug failures, find slow steps, and follow multi-turn conversations. They read the traces your agent sends. For the underlying concepts, see the section called “Monitor AI agents”.

- the section called “Agent traces” — inspect a single agent run, span by span.
- the section called “Agent sessions” — follow a multi-turn conversation across runs.
- the section called “Agent topology” — see how agents, tools, and models call each other.

Which view to start with

- You know which run failed. Open it in Agent traces and follow the spans.
- The problem is conversational (a user says the agent "forgot" or "went in circles"). Start in Agent sessions, which shows the agent's behavior across turns.
- You have a symptom but no specific run ("this tool fails often," "something got slow"). Filter by errors, latency, or token usage in Agent traces and Agent sessions to find the runs that show why. Use the agent topology to see how your agent's components connect.

These views focus on individual runs. For production and fleet-wide monitoring across your agents in the Omni web UI, see the section called “Monitor agents in the web UI”.

Agent traces

Use Agent traces to find the agent runs you need to inspect, open a run to see what the agent did, and act on what you find: score runs with evaluators, turn failures into dataset test cases, and compare runs side by side. A trace records one complete agent run; a span records one operation within it, such as a model invocation, a tool call, or a retrieval. For the full trace, span, and session vocabulary, see the section called “Monitor AI agents”.

For traces of requests crossing your services and dependencies, see the section called “Application traces”.

Open Agent traces

In the launcher, under Agent observability, choose `Agent traces`, then select the time range you want to investigate. The summary above the results shows totals for the selected range, such as Total traces, Traces with errors, duration, and token usage.

The IDE extension has the same view for the traces your agent produces while you develop. In local mode it reads the runs on your machine. See the section called “Develop with the IDE extension”.

Find the runs to inspect

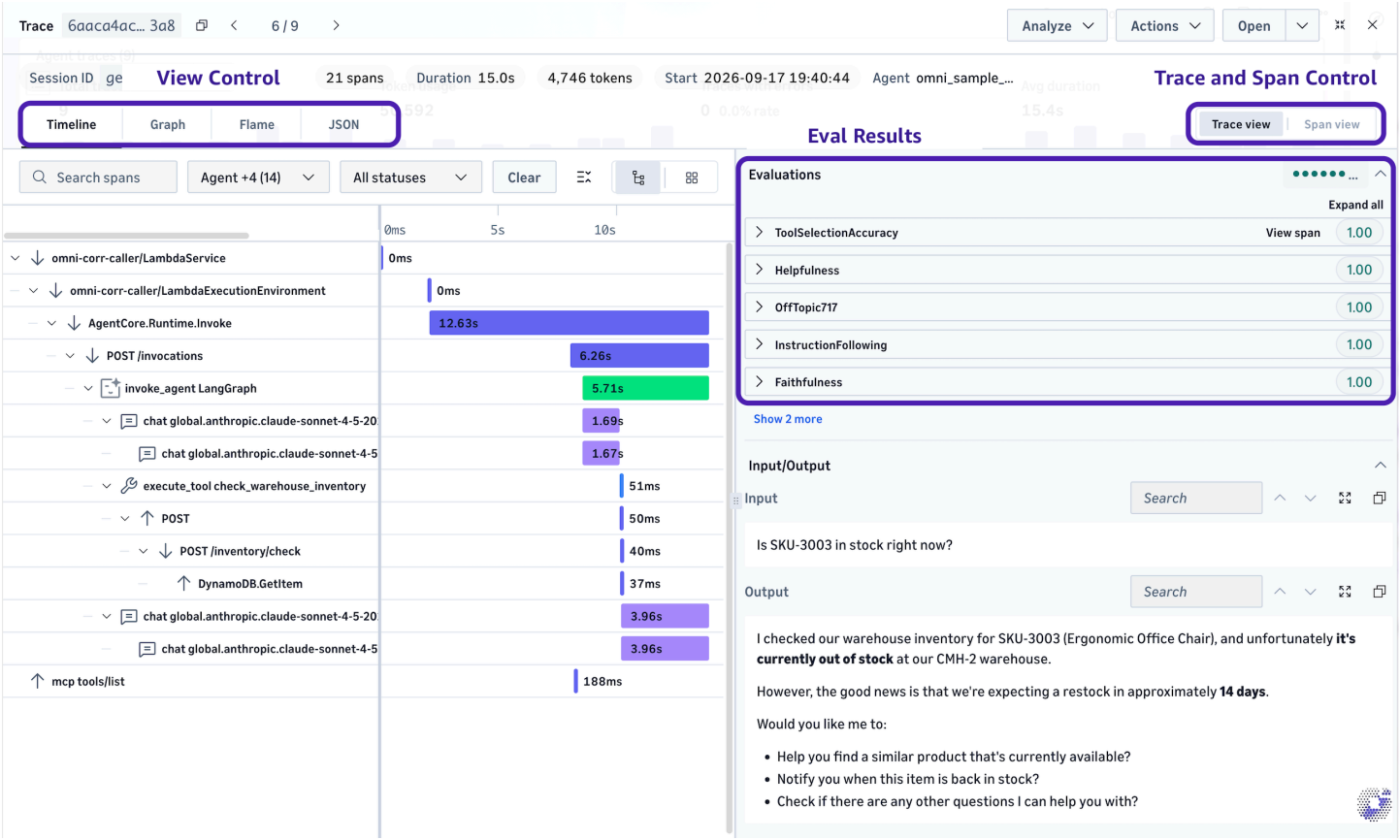
The traces list supports several ways to narrow the results:

- **Filter.** Use the filter panel to narrow by agent, status, latency, token usage, and other trace properties.
- **Filter by evaluation score.** Narrow the list to traces a specific evaluator scored above or below a threshold. For example, filter on a score range such as 0.0 to 0.5 to surface runs that completed without an error but failed evaluation. A high score range collects examples of the behavior you want to preserve.
- **Search.** The search box above the table narrows the traces already loaded, matching text anywhere in a trace ID, a run's input, or its output. It reports how many of the loaded traces matched, and highlights the matching text in them. It matches no other field and does not re-query your time range, so use the filter panel to narrow by anything else (an evaluator or its score, for example).

Trace context does not propagate by itself: both sides of each hop must be instrumented. If an agent's caller is not instrumented, the agent's runs still appear here. They start at the agent.

Inspect a run

Select a trace to open it. The header shows the trace ID and span count, and the run's spans open in execution order so you can follow what the agent did step by step.



Two controls change what the detail shows:

- **The view control** changes how the spans are drawn: a timeline of span bars to follow timing and parent-child structure, or a raw view of the span records.
- **The Trace and Span controls** change the scope of the detail panel:
  - **Trace** shows the whole run: the run's input and output, its status, latency, token usage, and its evaluation results.
  - **Span** shows one operation. Select a span to see the service that produced it, the span's own input and output, its timing and status, its attributes, and — for a tool call — the tool's parameters.

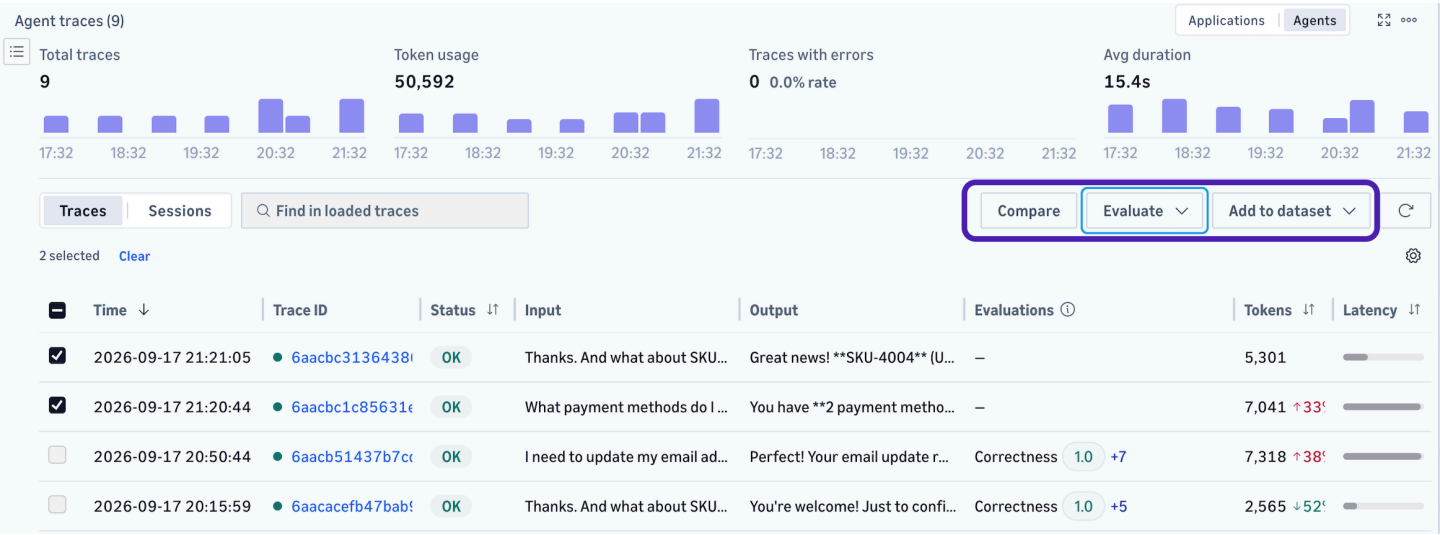
Reading evaluation results. The Trace scope lists every evaluation recorded for the run, each with the evaluator's name and score. Expand a result to read its explanation: the judge model's

reasoning for why the run received that score. Rule-based evaluators report the score alone, with no explanation. A result that scored one tool call links to that span, so you can jump from the score to the operation it judged.

Pivot to related telemetry. From an open run you can open the logs that carry the same trace ID (**Show logs for this trace**), open the agent's own metrics (**Show agent metrics**), and, when your space is connected to the AWS DevOps Agent, start an investigation from a failed run (**Investigate**). See the section called “Investigate with the AWS DevOps Agent”.

Act on what you find

Select one or more runs in the traces list, then choose an action:



- **Evaluate.** Pick the evaluators to run, and scores appear alongside the traces as they complete, each with its explanation on the trace. Two kinds of evaluator cannot run here. Session-level evaluators score whole conversations; run them from the section called “Agent sessions”. Evaluators that require an expected response need ground truth a live trace does not carry; run those against a dataset instead. See the section called “Evaluators and evaluations”.
- **Create dataset** or **Add to dataset.** Turn the selected runs into test cases: each run's input becomes the test input, and its response and tool calls seed the expected outcome for you to review. Add failing runs before you fix them so that the fix can be verified against a test case. See the section called “Datasets”.
- **Compare.** Select exactly two traces to see them side by side. Comparing a failing run with a successful one shows where the behavior diverged.

For the end-to-end workflow that runs through this page (find a failing run, package it, fix it, verify the fix), see the section called “Tutorial: Fix a production quality regression”.

When traces are missing or incomplete

What you see on this page depends on the telemetry your agent sends. These checks cover the common gaps:

- No traces appear. Confirm the time range, and verify that the agent is sending traces to your space. See the section called “Send AI agent telemetry”.
- A run has only one or a few flat spans. The agent's internal operations are not instrumented, so there is nothing deeper to inspect. See the section called “Send AI agent telemetry”.
- Inputs, outputs, or tool parameters are blank. The instrumentation does not capture that content. The symptom table in the section called “Send AI agent telemetry” maps each gap to the configuration that fixes it.
- Logs do not appear from the **Show logs for this trace** pivot. Confirm the agent's logs reach your space and carry the trace ID for correlation.

Where to go next

To do this	See
Follow a conversation across runs	the section called “Agent sessions”
See your agent's structure and where runs fail in it	the section called “Agent topology”
Score runs continuously instead of on demand	the section called “Evaluators and evaluations”
Manage the test sets you build from runs	the section called “Datasets”
Query trace records directly with SQL	the section called “Explore and query your telemetry”
Walk the find, fix, verify workflow end to end	the section called “Tutorial: Fix a production quality regression”

Investigate traces of requests across your services	the section called “Application traces”

Agent sessions

The Agent sessions view groups the traces that belong to the same conversation or user interaction, so you can follow a multi-turn exchange from start to finish and see where it went wrong across turns. For how traces, spans, and sessions relate, see the section called “Monitor AI agents”.

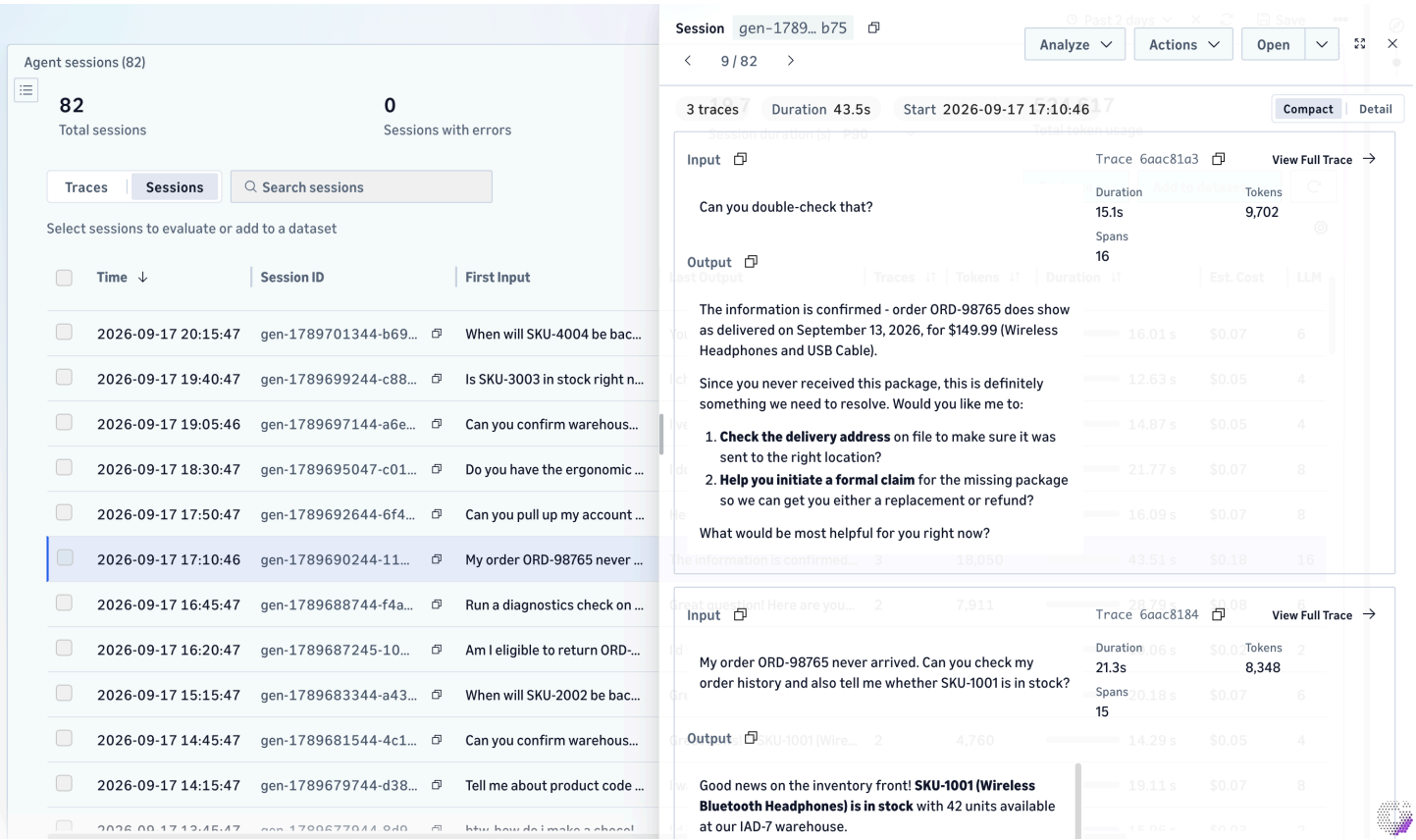
Why sessions need their own view

Some failures are invisible at the single-trace level because every individual turn looks fine. The Agent sessions view surfaces the cross-turn patterns:

- **Lost context** — the agent asks for information the user already provided, or contradicts an earlier answer.
- **Churn** — the same tool is called with the same arguments across consecutive turns, or the user rephrases the same request repeatedly (a sign the agent is not making progress).
- **Degradation over length** — quality that is fine on turn one and poor by turn eight, which single-trace sampling never shows.

To follow a session

1. Open your space, then choose `Agent sessions`.
2. Select a session to see its turns in order and read the conversation flow.
3. Select the turn where the problem starts to open its trace in the section called “Agent traces”.



Evaluate and reuse sessions

Cross-turn quality checks run from this view. Select one or more sessions, and then choose an action from the toolbar:

- **Evaluate** — score the selected sessions. Evaluation runs through Amazon Bedrock AgentCore as a background batch job, and scores appear alongside the sessions as they complete. Only evaluators that score whole sessions can run here; evaluators that score individual traces run from the section called “Agent traces”. For the evaluator catalog and how scoring works, see the section called “Evaluators and evaluations”.
- **Create dataset** or **Add to dataset** — turn the selected sessions into test cases, so a multi-turn scenario becomes a repeatable test. You choose whether each session becomes one test case or each of its traces does. See the section called “Datasets”.

Group traces into sessions

Omni groups traces into sessions using the OpenTelemetry `session.id` span attribute. Agents hosted on Amazon Bedrock AgentCore, and agents built with frameworks that set the attribute

natively (such as Strands Agents), are grouped automatically. Custom and non-framework agents must set the attribute themselves. Otherwise, each turn appears as an unrelated trace.

Set `session.id` on the root span of every turn that belongs to the same conversation, reusing one stable ID per conversation and starting a fresh ID when a new conversation starts:

```
from opentelemetry import trace

# Reuse the SAME id for every turn in one conversation.
session_id = "conv-8f2a1c"  # your stable per-conversation id

span = trace.get_current_span()
span.set_attribute("session.id", session_id)
```

Note Only `session.id` groups traces into sessions. Setting the `gen_ai.conversation.id` attribute is harmless (other tools read it), but Omni does not use it for session grouping.

Agent topology

The agent topology shows how the parts of your agent connect — agents, tools, and models — as a graph built from your traces. Where a trace shows one run in sequence, the topology aggregates the paths your agent took across runs: which components exist, which call which, and where latency concentrates.

The agent topology is in **preview release** and is subject to change.

For the map of your services and their dependencies, see the section called “Monitor applications with the application map”.

How the graph is built

Omni derives the graph from your spans: each agent, tool, and model that appears in your traces becomes a node, and the transitions between them (which component ran after which) become the edges. The graph is only as complete as your instrumentation: if tool calls are not instrumented as spans, those tools do not appear in the graph. See the section called “Supported environments, languages, and frameworks”.

Choose a view

The **Flow** and **Graph** toggle changes how the same nodes and transitions are drawn:

- **Flow**, the view the map opens in, reads left to right as the sequence of steps your agent ran, aggregated across runs.
- **Graph** draws the components and transitions as a network diagram. Use it to inspect the branching structure and the connections between components.

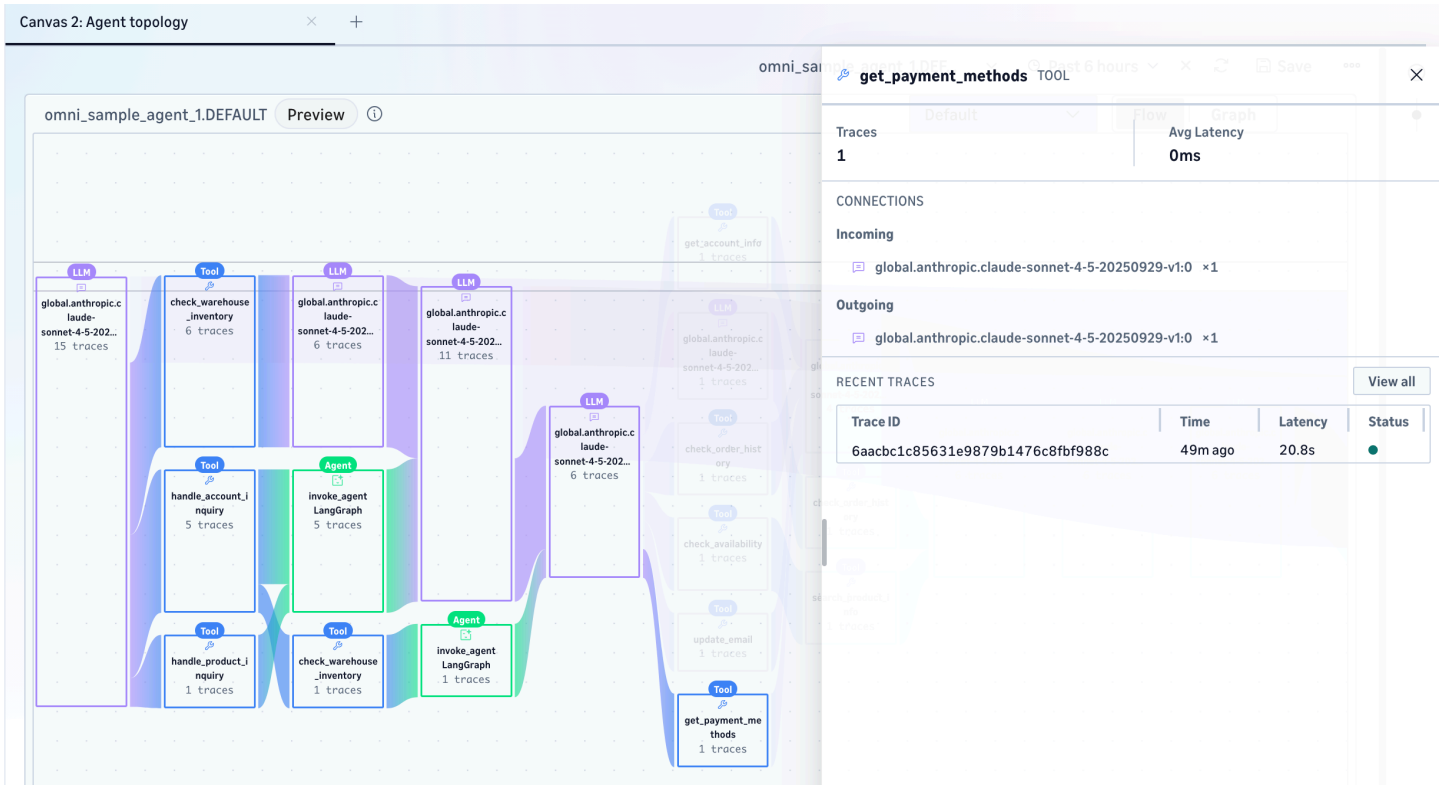
Group spans into nodes

The selector next to the view toggle offers two ways to group spans into the map's nodes:

- **Default** builds the map from the span kinds your instrumentation already emits: agent, tool, and model spans in the order they ran, with parallel calls grouped into a single step. It works with traces from any supported framework and needs no additional span attributes.
- **Steps** builds the map from explicit graph metadata on your spans: `agent_graph.node` names the node a span belongs to, `agent_graph.step` orders the nodes, and `agent_graph.kind` optionally sets a node's type. Step spans must be direct children of the trace's root span. LangGraph instrumentation adds this metadata automatically; for other frameworks, set the attributes yourself (for example, in a span processor). Choose **Steps** when your agent is authored as an explicit graph and you want the map to mirror the nodes you defined. If the selected traces carry no step metadata, the map reports it. Switch back to **Default**.

To view the agent topology

1. Open your space, then choose **Agent topology**.
2. Select a node to see its trace volume and average latency.
3. From a node with high latency, open its related traces to see those calls in context. See the section called “Agent traces”.



Evaluate agent quality

Agents fail in ways that do not show up as errors: the run completes, but the answer is wrong, incomplete, or against your policies. Evaluation is how you measure response quality, so you can catch these failures and tell whether a change to a prompt, a model, or a tool made your agent better or worse.

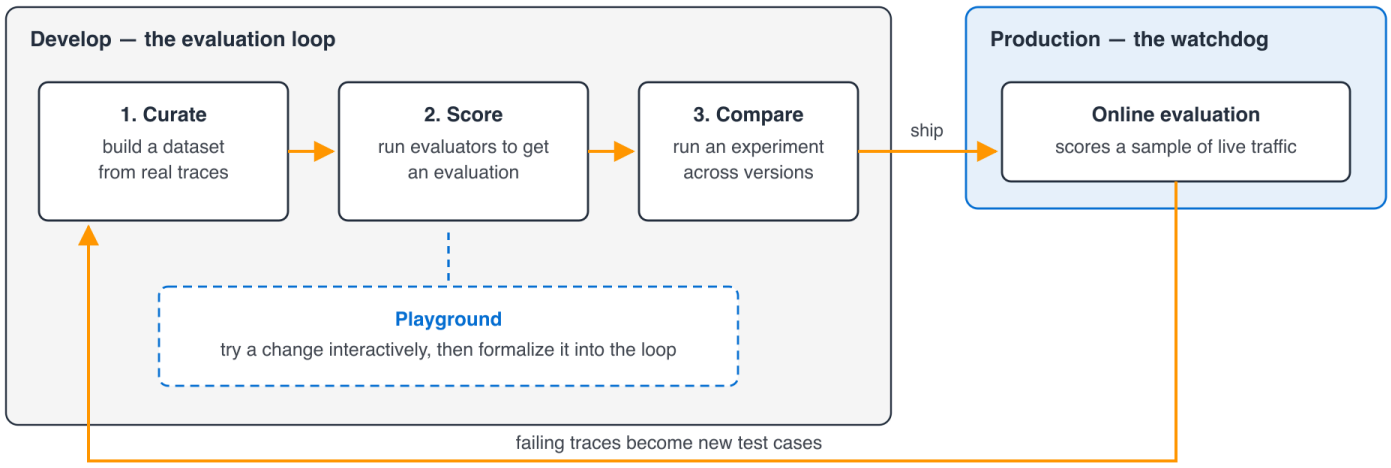
How evaluation fits together

The evaluation workflow builds from your real traffic:

1. **Curate test cases from real traces.** Your traces already contain real requests and the behavior that needs to improve. You collect representative runs (common requests, edge cases, known failures) into a **dataset**, a fixed test set you can run against repeatedly. See the section called “Datasets”.
2. **Score responses with evaluators.** An **evaluator** reads a trace (the input, the response, and the steps in between) and returns a score against a criterion such as correctness or relevance. The scores from running one or more evaluators are an **evaluation**, the record you compare against later runs. You can score a single trace as you debug, or score your agent's outputs across a whole dataset. See the section called “Evaluators and evaluations”.

3. **Compare versions with experiments.** An **experiment** runs two or more agent versions against the same dataset with the same evaluators, so a score difference can only come from the change you made. See the section called “Experiments”.

The **playground** sits alongside this loop for interactive iteration: try a prompt or an agent change and see the effect immediately, then formalize what worked into a dataset and an experiment. See the section called “Prompt playground”.



Where evaluators come from

Every evaluator runs through Amazon Bedrock AgentCore Evaluations, so one catalog covers three kinds:

- **Built-in** — ready-made checks for common criteria such as correctness, helpfulness, and tool-use quality. Start here.
- **Third-party** — metrics from the DeepEval and AutoEval open-source libraries, run for you, so you can use familiar metrics without hosting them.
- **Custom** — evaluators you write yourself, including one that reuses a built-in or third-party evaluator's scoring on a judge model you choose.

Choose an evaluator by the criterion you want to check; you can mix kinds in the same evaluation. For the full catalog and more on how to write your own evaluators, see the section called “Evaluators and evaluations”. To understand how Amazon Bedrock AgentCore evaluates agent

performance, see Evaluate agent performance and Third-party evaluators in the *Amazon Bedrock AgentCore Developer Guide*.

Evaluation during development and in production

The same pieces serve two workflows. During development, you run the loop deliberately: build a dataset, score a change in the playground, confirm it with an experiment before you deploy. In production, evaluation runs unattended: an **online evaluation** is a configuration that continuously scores a sample of your live traffic with the evaluators you choose (see the section called "Online evaluations"), and a declining pass rate shows that quality regressed. When production evaluation catches a regression, the development loop is how you fix it. See the section called "Tutorial: Fix a production quality regression" for the end-to-end workflow.

Evaluators read trace content, so your agent must send traces that include input and output. See the section called "Send AI agent telemetry". For the trace, span, and session vocabulary these pages build on, see the section called "Monitor AI agents". To monitor quality signals across your agents in production, see the section called "Monitor agents in the web UI".

Evaluators and evaluations

An evaluator scores agent behavior against a criterion (for example, whether the response was correct, relevant, or grounded in the retrieved context). An evaluation is the result of running one or more evaluators over a trace, or over the outputs your agent produces from a dataset's test cases. Because evaluators are automated, you can score hundreds of runs the way you would review one by hand. This page covers scoring on demand. To score production traffic continuously, see the section called "Online evaluations".

What an evaluator reads

An evaluator reads the full trace — the user's input, the agent's response, and depending on the evaluator, the steps in between: which tools the agent called and in what order, and what context it retrieved (for the trace and session vocabulary, see the section called "Monitor AI agents"). Each evaluator scores at one of three levels: a **trace** (one request and response), a **session** (a whole conversation), or a **tool call** (one tool invocation). Traces must therefore show input and output. If your evaluators return blank results, check the content-extraction settings in the section called "Send AI agent telemetry".

Where evaluators come from

Every evaluator runs through Amazon Bedrock AgentCore Evaluations, and the catalog covers three kinds. For the full evaluator reference, see Evaluate agent performance in the *Amazon Bedrock AgentCore Developer Guide*.

- **Built-in** — ready-made checks for common criteria such as correctness, helpfulness, and tool-use quality. Start here: they cover the criteria most agents need, without setup. Some built-in evaluators use a judge model internally; others are rule-based checks.
- **Third-party** — metrics from the DeepEval and AutoEval open-source evaluation libraries. If you already use these libraries, you can run the same metrics here without hosting them: the service runs the evaluator, selects the judge model, and manages the library version, the same way it does for a built-in evaluator. To run one of these metrics on a judge model you choose, create a derived custom evaluator instead. DeepEval covers task completion, tool use, and summarization; multi-turn behavior such as knowledge retention, turn relevancy, and goal accuracy; and safety checks for bias, toxicity, and PII leakage. AutoEval covers whether a response is malicious, whether it is humorous, and whether the agent attempted the task or declared it impossible. AWS tests and benchmarks the built-in evaluators but makes no quality claims about these libraries, so validate a metric against your own traces before you rely on its scores. For the metric-by-metric list, see Third-party evaluators in the *Amazon Bedrock AgentCore Developer Guide*.
- **Custom** — evaluators you create, in three forms. A **model-as-judge** evaluator scores trace content against criteria you write in natural language, with your choice of judge model. Use it for criteria that need judgment ("is the tone appropriate for a support agent?") that no fixed rule can check, and spot-check its scores against your own judgment before trusting it at scale. A **derived** evaluator runs an existing built-in or third-party evaluator's logic on the judge model you choose, so you skip writing instructions and a rating scale. A **code-based** evaluator is your own AWS Lambda function that receives the trace content and returns a score. Use it for deterministic, domain-specific checks (the response parses as valid JSON, cites a real order ID, stays under a length limit) where a rule is cheaper and more reliable than a model.

> **Important**
>
> **Judge-based evaluators send trace content to the judge model.** Both the evaluator prompt and the trace content it scores go to the model for inference. This applies to judge-

based built-in and third-party evaluators as well as custom model-as-judge evaluators. If your traces can contain sensitive data, see the section called “Protect sensitive data”.

Permissions for custom evaluators

Built-in and third-party evaluators run on service-managed capacity — there is nothing to grant. A custom evaluator invokes its judge using your space's permissions, so the Space Access role must be allowed to invoke it, and the permission depends on the judge:

- Amazon Bedrock foundation model or inference profile — `bedrock:InvokeModel` and `bedrock:InvokeModelWithResponseStream` on that model or inference profile.
- A model served through the `bedrock-mantle` endpoint in Amazon Bedrock — `bedrock-mantle:CreateInference` on the project and `bedrock-mantle:CallWithBearerToken`.
- Lambda (code) evaluator — `lambda:InvokeFunction` and `lambda:GetFunction` on your evaluator function.

Without the matching permission, on-demand scoring with that custom evaluator fails; built-in and third-party evaluators are unaffected. For the managed policies that carry these actions and the conditions on them, see the section called “IAM policies to use CloudWatch Omni”. For online evaluations, grant them to the evaluation's execution role instead — see the section called “Online evaluations”.

Built-in evaluators miss domain-specific failures

Built-in evaluators score general criteria, and an agent can pass all of them while doing the wrong thing for your domain. Consider a customer support agent that answers an off-topic question, such as providing a recipe: a helpfulness evaluator scores it high (it answered well), and a policy-compliance evaluator scores it high (nothing harmful). Every score is high, yet the agent has left its designated scope. No general-purpose criterion knows what your agent's scope is.

Use built-in evaluators for universal criteria, and write a custom evaluator for every rule that is specific to your agent: permitted topics, required behaviors, forbidden actions. If you can state the rule in a sentence, you can usually write it as a model-as-judge evaluator; if it is mechanically checkable, write it as a code-based evaluator.

Write a custom model-as-judge evaluator

A model-as-judge evaluator is a prompt to the judge model plus a rating scale. The judge reads the conversation and the response, applies your rubric, and returns one of the scale's values. A well-specified evaluator has three parts:

1. **State what the agent is, and what is in scope.** Enumerate the permitted topics or behaviors explicitly. The judge cannot infer your product's boundaries. For example: "The agent is an e-commerce customer support agent. Its only permitted topics are returns and refunds, product inquiries, and account management."
2. **Write the rubric as decidable cases.** Define each score with the specific conditions that earn it, including the edge cases. For a scope evaluator: "a response that acknowledges the request is off-topic but still answers it is off-topic." Vague rubrics ("score how appropriate the response is") produce inconsistent judgments.
3. **Match the rating scale to the criterion.** A rule the agent either follows or breaks should be binary: two values, pass and fail, each with a one-line definition. Remove middle values; a 0.5 on "did the agent stay in scope?" has no meaning, and evaluators default to it when the rubric leaves room for doubt. Reserve graded scales for criteria that are genuinely graded, such as completeness.

The evaluator's instructions include placeholders that are filled with trace content for each run it scores. For a trace-level evaluator, these are `{context}` (the conversation so far) and `{assistant_turn}` (the response under evaluation). Each evaluation level has its own placeholder set; for the full list, see Create evaluator in the *Amazon Bedrock AgentCore Developer Guide*. The rating scale can be numerical (each value has a number, a label, and a definition) or categorical (the label is the value). Test the evaluator on a known-bad and a known-good trace before relying on it: the known-bad trace should score fail, and the known-good one should score pass. If either misses, tighten the rubric.

To evaluate a response

1. Open a trace in the section called “Agent traces” to score a single run, select sessions in the section called “Agent sessions” to score whole conversations, or run your agent against a dataset's test cases, for example in an experiment, and score the outputs. See the section called “Experiments”.
2. Choose the evaluators that match what you want to check: one criterion per evaluator, and typically several evaluators per run.

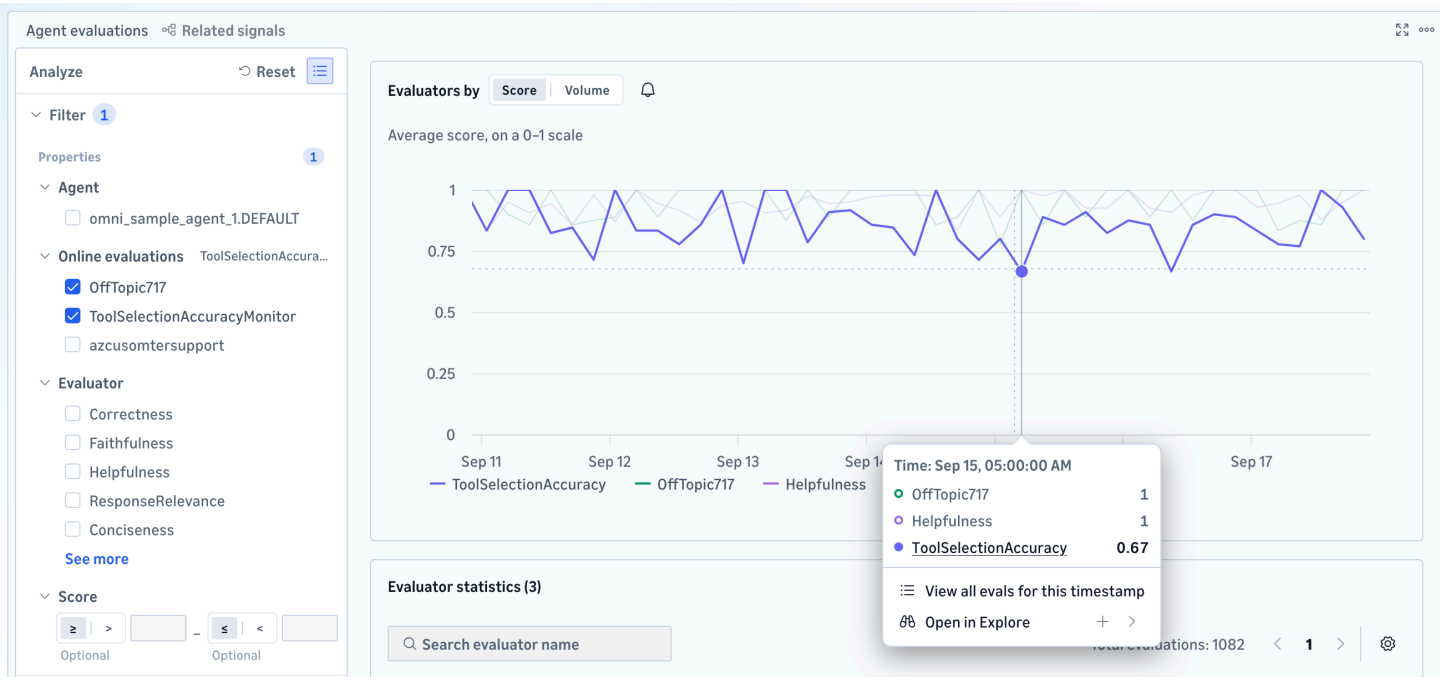
3. Run them, and review the scores. Each score links back to the trace it came from, so you can open a low-scoring run and see exactly what the agent did.

Interpret scores as comparisons

A single absolute score rarely means much on its own. A 0.8 correctness score is only good or bad relative to something. Scores become useful when you compare: the same dataset before and after a change (see the section called “Experiments”), the same evaluator across agents, or a trend over time.

Review evaluation results over time

The **Evaluation dashboard** aggregates your evaluation results: score trends, score distributions, pass rates, and a comparison across evaluators. Use it to watch for drift after individual runs are scored: a pass rate that declines over days signals a regression even when every individual score looks reasonable in isolation. From a declining trend, open the low-scoring traces to see what changed. If the drop follows a deployment or a prompt change, check that change first. To respond to a regression you find here, see the section called “Tutorial: Fix a production quality regression”.



Datasets

A dataset is a curated collection of test cases that you evaluate against repeatedly. It gives you a repeatable measurement: because the test cases stay fixed, you can run them after every change

and compare scores over time. These evaluation datasets are distinct from the CloudWatch Dataset, the store that holds your space's telemetry (see *CloudWatch Omni*).

What a test case contains

Each test case in a dataset represents one scenario:

- **An input** — the request the agent should handle. For a multi-turn scenario, a test case carries the conversation turns, which run in order within a single session so the agent keeps context between them.
- **An expected outcome (optional)** — a reference response, the tools the agent is expected to call, or natural-language assertions the response must satisfy. Evaluators that compare against a reference need this; evaluators that judge the response on its own do not. Every test case follows the same dataset schema, so any evaluator can draw the fields it needs from any case.

Build from real traces

You can write test cases by hand, but start from real traffic. Your traces already contain the requests your users make and the failures they hit. When you add a trace to a dataset, Omni converts it into a test case: the trace's input becomes the test input, and its response and tool calls can seed the expected outcome for you to review and edit.

To build a dataset from traces

1. In the section called “Agent traces” or the section called “Agent sessions”, find representative runs. Aim for a mix: common requests that must keep working, edge cases, and failures you have fixed and do not want to regress.
2. Add each run to a dataset. Review the generated test case, and record or correct the expected outcome where your evaluators need one.
3. Run your agent against the dataset's test cases (for example, in an experiment) and score the outputs with evaluators to get a baseline. See the section called “Evaluators and evaluations” and the section called “Experiments”.

Publish a version before you use it

A dataset you are editing is a working draft. Before you can run it (in the prompt playground or an experiment), you publish it as a numbered version. Publishing matters because:

- **A version is immutable.** Once published, its test cases are frozen, so a score is always tied to an exact set of cases. Continued editing happens on the draft and does not disturb versions you have already run.
- **Scores stay comparable over time.** Comparing two agent versions, or a before/after change, is valid only when both ran against the same fixed cases. Versioning guarantees that.
- **You can keep publishing.** As you add or refine cases, publish a new version. Earlier versions remain available, so you can always re-run an old baseline or track which version produced which result.

Keep the dataset honest as your agent evolves

- When you find a new failure in production, add its trace to the dataset before you fix it. The fix then has a test case proving it. Publish a version so the fix is measured against a frozen baseline.
- Keep failure cases in the dataset after fixing them; they are your regression suite.
- If your agent's scope changes, retire test cases that no longer represent real traffic. A dataset full of obsolete cases rewards the wrong behavior. Publish a fresh version once you have pruned.

Use a dataset in the prompt playground

In the the section called “Prompt playground” you can run your baseline and candidate variants against a published dataset instead of one hand-typed input, so a comparison covers your whole curated set at once. A candidate that passes cases the baseline fails is evidence that the change works. See the section called “Prompt playground”.

Datasets move between surfaces

Datasets sync in both directions between the Omni web UI and the IDE extension, using the **Upload to Cloud** and **Download to Local** buttons:

- **Download to Local** brings a cloud dataset into the extension, so you can run it against your local development loop.
- **Upload to Cloud** pushes a dataset you built locally (for example, while running your agent on your own machine) up to the Omni web UI, where the team can use it for experiments and shared review.

An operator packages failing traces into a dataset in the Omni web UI, and a developer downloads it to reproduce and fix the problem locally. In the other direction, a developer curating cases against a local agent uploads them so the same cases are available in the cloud. For the end-to-end workflow, see the section called “Tutorial: Fix a production quality regression”.

Each transfer creates a new dataset. Upload to Cloud and Download to Local always create a **new** dataset in the target environment. They do not publish a new version of an existing dataset, and they do not merge into an existing one.

Experiments

An experiment tells you whether a change made your agent better or worse before you ship it. It works by controlling the variables: the same dataset and the same evaluators run against two or more agent versions, so a difference in scores can only come from the change you made.

Available in the IDE extension only.

What you can compare

A "version" in an experiment is any variant of your agent you can run: a new prompt version, a different model, a changed tool, or a reworked orchestration step. Change one thing per experiment where you can. If a variant changes the prompt and the model at once, the scores cannot tell you which change mattered.

To run an experiment

1. In the IDE extension, choose a dataset to run against. Use one with a baseline score, so you know what you are comparing to. See the section called “Datasets”.
2. Select the agent versions or configurations to compare, and the evaluators to apply. Keep the evaluators identical across versions.
3. Run the experiment. Each version runs every test case in the dataset, and the evaluators score each run.
4. Compare the results: the aggregate scores first, then the per-case results.

Read the per-case results, not only the average

An aggregate score can hide a trade-off: a variant that improves the average while breaking cases that used to pass may be worse than the number suggests. Look at the cases that changed direction (improved and regressed) and open the traces behind the regressions to see what the new

version did differently. Add a regressed case to your dataset so that later versions are measured against it. See the section called “Datasets”.

Online evaluations

On-demand evaluation scores a trace when you ask. An **online evaluation** scores your traffic continuously: you choose the agents to watch, the evaluators to run, and how much traffic to sample, and CloudWatch Omni evaluates matching traces as they arrive. Quality regressions surface in your evaluation trends without anyone running a check. Online evaluation is the continuous-monitoring stage of the evaluation workflow described in the section called “Evaluate agent quality”.

How an online evaluation works

An online evaluation is a configuration that you create, and then enable:

- **Agents to evaluate** — chosen by service name from the agents seen in your space's telemetry.
- **Evaluators to run** — one or more evaluators from the catalog described below.
- **Sampling percentage** — how much of the matching traffic is evaluated. Sampling is your cost control: judge-based evaluators invoke a judge model for every sampled trace, so evaluate a fraction of high-volume traffic.
- **Session settings (optional)** — a session timeout that determines when a conversation is considered complete, so session-level evaluators can score the full exchange. The timeout is measured from a session's last activity: a session is evaluated only after it has been idle for this long. A session that never goes idle — for example a long-running or continuously reused session that always has recent activity — is never treated as complete, so it is never evaluated and produces no results. Set the timeout shorter than the typical gap between turns in your sessions so they finalize and are scored.

While the configuration is enabled, CloudWatch Omni evaluates sampled traces continuously. Scores appear alongside the traces and feed the trends in the Evaluation dashboard. Each configuration also gets a dedicated CloudWatch log group holding its results in JSON, and publishes each score as a CloudWatch metric, so you can query results with CloudWatch Logs Insights and alarm on a score the same way you alarm on any other metric. Results follow the OpenTelemetry semantic conventions for generative AI evaluation events, and each result references the trace and session it came from, so you can go from a score back to the run that produced it.

To change an enabled configuration's evaluators, disable it first. The evaluator list is locked while the evaluation runs. A running evaluation also locks the custom evaluators it uses, so that scores stay comparable over time: you cannot edit or delete one while an evaluation is using it. To change a custom evaluator's criteria, clone it, edit the copy, and point the configuration at the copy. The original keeps the scoring history it already produced.

Where evaluators come from

An online evaluation runs the same catalog as on-demand evaluation (built-in, third-party, and custom evaluators, all through Amazon Bedrock AgentCore Evaluations), and you can mix kinds in one configuration. For what each kind covers and how to write your own, see the section called "Evaluators and evaluations".

The judge model depends on the evaluator kind, and that determines what you pay for. For built-in and third-party evaluators, the service selects and runs the judge model on its own capacity. There is nothing to configure. For custom evaluators, including one derived from a built-in or third-party evaluator, the judge model is the one you chose, and it is invoked with the permissions you grant the evaluation, so those model invocations run in your account. For the online-evaluation reference, see Evaluate agent performance, Online evaluation configurations, Prerequisites, and Results and output in the *Amazon Bedrock AgentCore Developer Guide*.

Permissions for custom evaluators

Built-in and third-party evaluators run on service-managed capacity — you grant nothing extra. A custom evaluator invokes the judge you chose, using the online evaluation's execution role, so that role must be allowed to invoke it. The permission depends on the judge:

- Amazon Bedrock foundation model or inference profile — `bedrock:InvokeModel` and `bedrock:InvokeModelWithResponseStream` on that model's or inference profile's ARN.
- A model served through the `bedrock-mantle` endpoint in Amazon Bedrock — `bedrock-mantle:CreateInference` on the project and `bedrock-mantle:CallWithBearerToken`.
- Lambda (code) evaluator — `lambda:InvokeFunction` and `lambda:GetFunction` on your evaluator function.

The default execution role created for you covers built-in and third-party evaluators. When you add a custom evaluator, add the matching permission above to the execution role before you enable the evaluation; otherwise enabling fails with a permissions error (for example, "the execution role does not have permissions to invoke the specified Bedrock models"). For the managed policies

that carry these actions and the conditions on them, see the section called "IAM policies to use CloudWatch Omni". For the execution-role policy reference, see Prerequisites.

To create an online evaluation

1. In your space, under `Evaluate`, choose `Online evaluations`, and then choose `New evaluation`.
2. Select the agents to evaluate, the evaluators to run, and the sampling percentage.
3. Grant the permissions the evaluation needs. Every online evaluation needs permission to write its results (CloudWatch Logs and metrics). If you include a custom evaluator, the evaluation's execution role must also be allowed to invoke that evaluator's judge — the exact permission depends on the judge's type (see **Permissions for custom evaluators**). Built-in and third-party evaluators need no additional permissions.
4. Enable the evaluation, and confirm its status shows as enabled. Scores from sampled traffic appear in the Evaluation dashboard as they accumulate.

Cost and data considerations

- Judge-based evaluators invoke a model for every sampled trace. Tune the sampling percentage to match your traffic volume and budget.
- Evaluators that use a judge model send the sampled trace content to that model. If your traces can contain sensitive data, see the section called "Protect sensitive data".

For scoring a specific trace or dataset on demand, see the section called "Evaluators and evaluations". When an online evaluation surfaces a regression, follow the section called "Tutorial: Fix a production quality regression" to fix and verify it.

Prompt playground

The Prompt playground is an interactive space to try changes and see the effect immediately, without deploying your agent. Where datasets and experiments give you repeatable measurement, the playground is for quick iteration: form a hypothesis, try it, and see the result. Then formalize what worked into a dataset case or an experiment.

Differences by surface In the Omni web UI, the playground supports **prompt testing** — try a prompt against a model and compare outputs. In the IDE extension, it adds **agent behavior testing**

— run your agent end to end against test inputs. Agent behavior testing is available in the IDE extension only.

Model access (first-time use) The playground invokes Amazon Bedrock foundation models directly. The first time you use a given model in your account, you may need to request access to it in the Amazon Bedrock console first — see Add or remove access to Amazon Bedrock foundation models. Until access is granted, runs against that model fail with an access error.

Work in variants: baseline against candidate

The playground is organized around side-by-side variants. You start with a baseline (your current prompt or agent configuration, unchanged) and add a variation that carries your proposed change, such as an appended instruction or a different model. Both variants run the same input, so the difference in output can only come from your change. Leave the baseline unchanged: if you edit both variants, you can no longer tell which edit caused the improvement.

Prompt testing

Prompt testing runs a prompt directly against a model — no agent in the loop — so you can isolate the prompt as the variable. Use it to compare phrasings, instructions, or models side by side before changing your agent.

1. Choose `Prompt playground`, and select a prompt version from your managed prompts as the baseline, or enter a prompt directly.
2. Add a variation, and apply your change to it. Leave the baseline unchanged.
3. Run both variants against the same input and compare the outputs. To score the comparison, run an evaluator over both results: a failing baseline and a passing candidate is evidence that the change works. See the section called “Evaluators and evaluations”.
4. When a variant performs better, save it as a new prompt version with a note describing the change, so your agent can use it and you can trace which version produced which behavior. See the section called “Manage prompts”.

Agent behavior testing (IDE extension)

Agent behavior testing runs your agent (its prompts, tools, and orchestration together) against test inputs from inside the extension. Unlike prompt testing, this exercises the full execution path, so you see the trace the run produced and can tell whether a change fixed the behavior or only moved

the failure. It uses the local dev server, so iterations stay on your machine. See the local testing steps in the section called “Develop with the IDE extension”.

You can also run the variants against the test cases in a dataset (in prompt testing or agent behavior testing), so the comparison covers your curated scenarios instead of a single hand-typed input. A dataset alone is not enough: the run requires a prompt version. The dataset supplies only the test cases. This is the step before formalizing the change as an experiment. See the section called “Datasets”.

Monitor agents in the web UI

Use the Omni web UI to monitor your agents in production, including individual agent traces and sessions, aggregate metrics such as request volume, errors, latency, and token usage, and dependency relationships through the application map (see the section called “Monitor applications with the application map”).

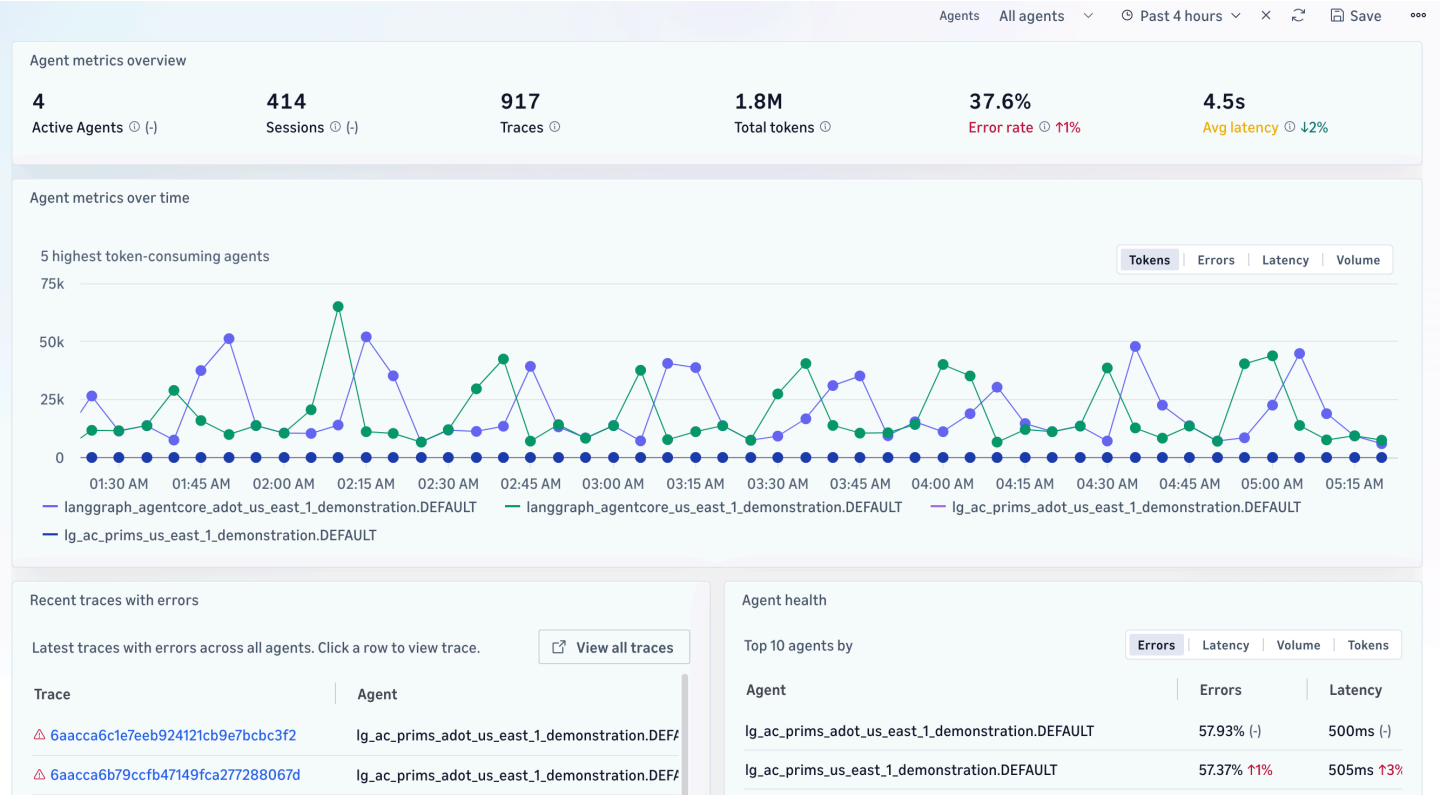
Aggregate views are available in the Omni web UI only. For monitoring and troubleshooting during development in the IDE, see the section called “Develop with the IDE extension”.

Aggregate metrics

- **Request volume** — the number of times the agent is invoked.
- **Errors** — the number of traces that contain errors.
- **Latency** — the duration of each run.
- **Token usage** — the total input and output tokens consumed.

To monitor your agents

1. Sign in to your space. To set up users and permissions, see the section called “Set up Omni”.
2. Review the aggregate views for request volume, errors, latency, and token usage across your agents. Filter by agent or time range.
3. Select a metric or a spike to view the underlying traces in the section called “Agent traces”, or follow a multi-turn conversation in the section called “Agent sessions”. If you do not yet know which runs to inspect, use the agent topology to see which components take part. See the section called “Agent topology”.
4. Check your evaluation results for the same period. To find evaluation trends, see the section called “Evaluators and evaluations”.



Combine these views with evaluation results. An agent can be fast, low-cost, and error-free but still give wrong answers. To measure response quality, see the section called “Evaluate agent quality”. When you find a quality regression here, follow the section called “Tutorial: Fix a production quality regression”.

Develop with the IDE extension

Use the CloudWatch Omni extension to build, test, and evaluate your agent from your IDE. The extension is available for VS Code, Kiro, and Cursor.

Local mode and cloud mode

The extension works against two data sources.

Mode	What you see	What it needs
Local	Traces from the agent running on your machine, stored in your project folder	Credentials for the model provider your agent calls. No AWS account connection

Mode	What you see	What it needs
Cloud	Traces from deployed agents — the same data your team sees in the Omni web UI	Sign-in to your space

Switch modes as you work. Build and test in local mode. Then switch to cloud mode to compare a local run against production traces, or to use a dataset that an operator packaged from those traces. See the section called “Tutorial: Fix a production quality regression”.

Features that interact with your project's code (such as the section called “Manage prompts”) are available in local mode only. If a feature needs the other mode, the extension shows it as unavailable and names the mode it requires.

Install and connect

1. Install the CloudWatch Omni extension in your editor. Search for it in your editor's extensions view, or install it from the registry your editor reads: Install for VS Code from the VS Code Marketplace, or Install for Kiro and Cursor from Open VSX.
2. Open your agent project. The extension creates an `.omni` folder in the project for local traces and configuration.
3. Choose local mode, or sign in to your space to use cloud mode. A space is required only for cloud mode. To create one, see the section called “Set up Omni”.

The extension sidebar shows the mode you are in and the data source it reads from.

Onboard your project

Onboarding instruments your project and connects the extension to your running agent. A coding agent in your editor does the work, using the Omni skills.

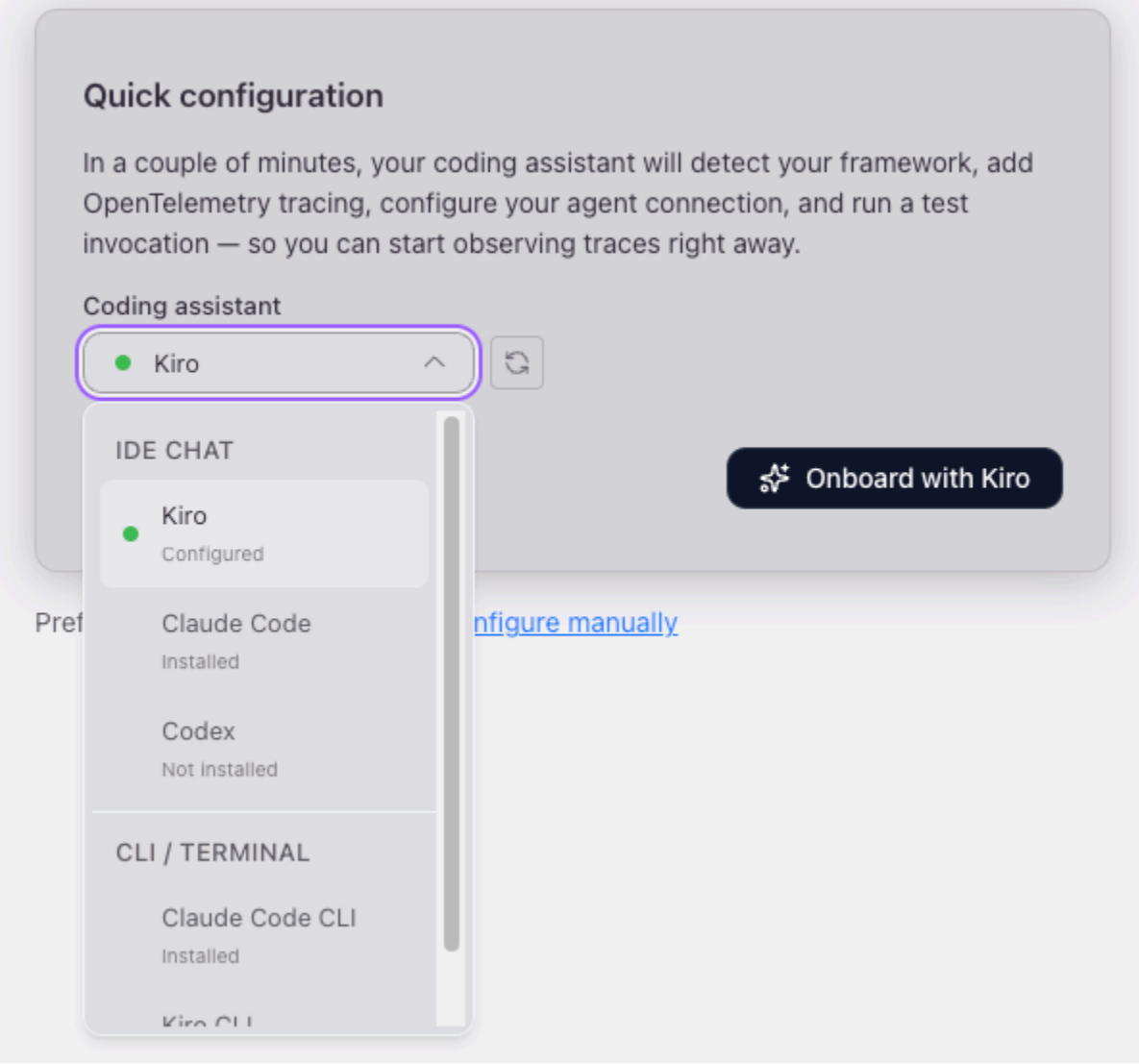
1. **Choose the coding agent you want to work with.** The extension installs the Omni skills that agent needs, and its guided flows run through that agent.
2. **Detect the framework and language.** The coding agent reads your dependency files and source imports to identify your agent framework and the language it is written in.
3. **Instrument the project.** The coding agent adds the OpenTelemetry instrumentation your agent needs to emit traces, choosing between AWS Distro for OpenTelemetry and OpenInference

according to your framework. Where instrumentation already exists, it verifies that setup rather than replacing it. For the instrumentation itself, see the section called “Send AI agent telemetry”.

4. **Connect to your agent.** The coding agent sets your start command and port and starts the local server, detects your agent's request and response schema from your code and saves it, then sends a test request. Onboarding is complete when that request produces a trace.

Start onboarding from the extension's Getting Started view.

Onboard manually instead. A coding agent is not required. Each step is also available on its own from the configuration view, so you can set the start command, the port, and the schema yourself, and re-run a single step without repeating the others. The "Test your agent locally" section of this page walks through the start command, the server, and the test request by hand.



Test your agent locally

Run your instrumented agent from inside the extension and inspect the traces it produces. If your agent is not instrumented yet, see the section called “Send AI agent telemetry”.

1. In the extension configuration, set the command that starts your agent's server and the port it listens on. The default port is 8080.
2. Start the server. The extension starts a local collector to receive telemetry, sets the OpenTelemetry environment your agent needs, runs your start command, and waits for the port to respond.
3. Send your agent a request (from the extension, from the playground, or from your own test harness) so the invocation produces traces that reach the collector.
4. Open `Agent traces` in the extension to inspect the run. Change your code or prompts and send another request to compare.

In local mode nothing leaves your machine: the collector writes traces to the `.omni` folder in your project.

Run your own server

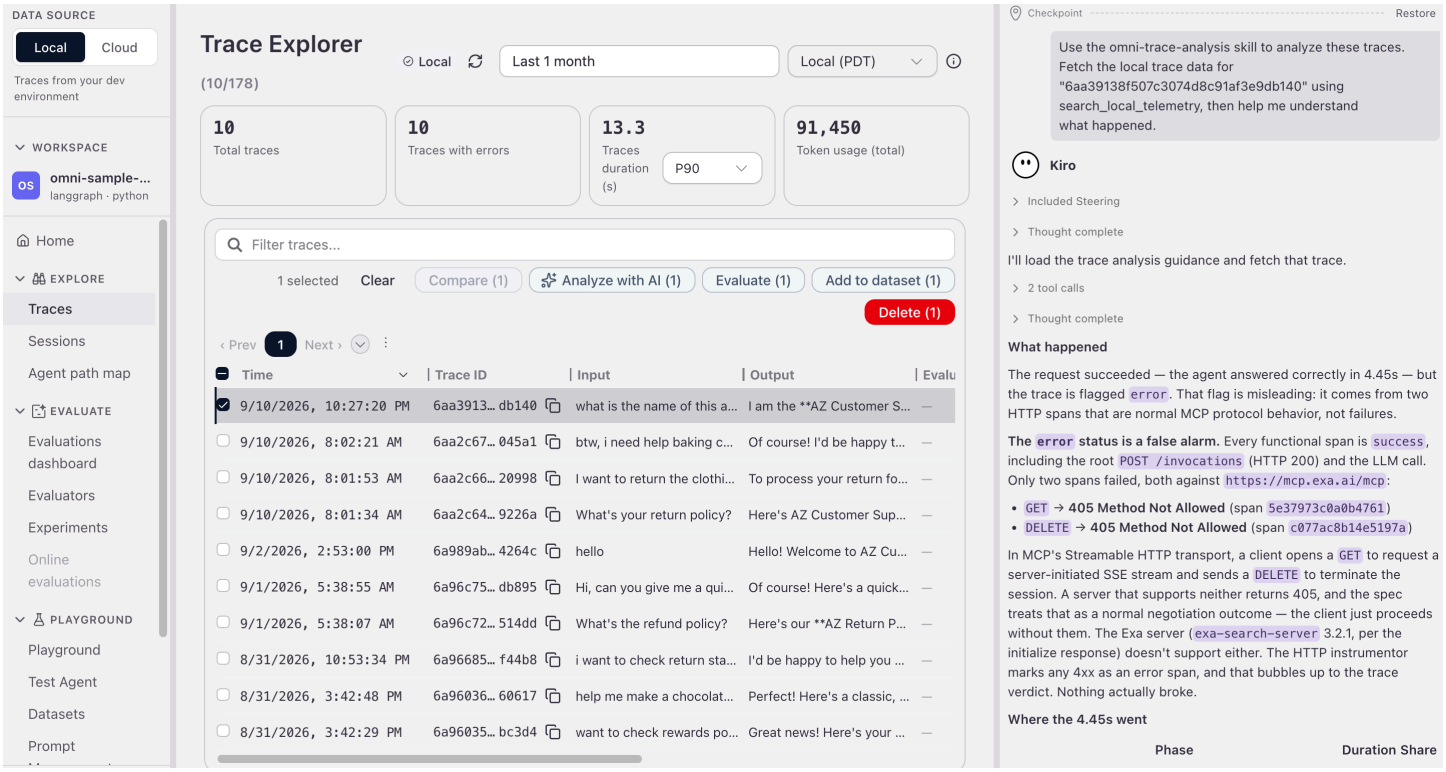
To start your agent yourself, switch the extension to external server mode. The extension still runs the collector. It writes the OpenTelemetry settings your server needs to a file in your project. Load that file before you start the server. If your server already listens on the configured port, the extension uses it and does not start a second collector.

AI assistance in the extension

The extension includes AI assistance for working on your agent:

- **Analyze with AI** — ask about a trace or a failure and get an explanation based on your telemetry.
- **Prompt recommendations** — get suggestions to improve a prompt.

You can use AI coding agents through the Omni skills in the Agent Toolkit for AWS. See the section called “Use Omni skills”. For the AI assistant in the Omni web UI, see the section called “Ask the Omni agent”.



Where to go next

Continue with these related topics:

- the section called “Manage prompts” — version your agent's prompts in the project and test changes before you ship them.
- the section called “Use Omni skills” — work on your agent from the coding agent you already use, through the Omni skills in the Agent Toolkit for AWS.
- the section called “Send AI agent telemetry” — instrument your agent so its runs produce the traces the extension reads.
- the section called “Agent traces” — how to read a trace, in the extension or the Omni web UI.

Manage prompts

Manage your agent's prompts as versioned artifacts in the extension, so you can track changes, compare versions, and reuse prompts across runs.

Availability Available in the IDE extension only.

You can also use and compare prompt versions in the prompt playground and in experiments. See the section called “Prompt playground” and the section called “Experiments”.

Bring existing prompts under management

Most agents start with prompts embedded in code. When you set up prompt management, your coding agent finds the prompts in your project and extracts them into managed, versioned artifacts. Each one becomes a named entry with a version history, and your code references the managed prompt instead of an inline string. From then on, every change is a recorded version.

To set it up:

1. In the IDE extension, open the prompt management view.
2. Follow the on-screen onboarding instructions.
3. When the flow hands off to your coding agent, invoke it to perform the extraction.

What a version captures

Each version records the prompt's messages and its model configuration (the provider, model, and inference parameters such as temperature and maximum tokens), along with a note describing the change. A comparison between versions reflects both. A version does not capture your agent's tools, orchestration, or code; to compare those, run an experiment. See the section called “Experiments”.

Versions are saved to a prompt store in your project, and each save is recorded as a git commit using your note as the commit message. Your prompt history is part of your project's history.

To create a new version

You create new versions from the playground: edit the prompt there, and when a change works, save it as a new version with a note describing the change. See the section called “Prompt playground”.

To review and manage versions

1. In the extension, open the prompt management view and choose a prompt.
2. Select a version from the timeline to see the messages and model configuration it recorded, and compare it with other versions.

3. Each version also shows the traces your agent produced while that version was in effect, so when behavior shifts you can tie the change to a specific version.
4. To try the selected version, choose `Test in Playground`. To roll back to it, choose `Revert`.

Reverting does not delete later versions. It creates a new latest version that restores the older version's messages and model configuration, so the rollback is itself a recorded change.

Monitor applications with the application map

Use the application map to understand how your applications and services interact with each other and to identify what needs the most attention. The application map shows your application services, their dependencies, and related cloud resources. You can also organize your application map by other dimensions such as environment, resource type, or related services.

Omni builds the application map from your telemetry for the time range you select. It then adds the infrastructure resources that it finds in your account. Omni observes service calls and resource access. It draws each one as an arrow between two nodes. When the map is grouped by application, Omni groups services by the OpenTelemetry `service.namespace` attribute. The `service.name` attribute names each service in that application. Services that do not set the `service.namespace` attribute are grouped under **None**. Services running in other clouds appear alongside your AWS services when their telemetry carries the standard OpenTelemetry resource attributes, and calls between them are drawn like any other.

You can access the application map from the Quick Launch menu in the Omni web UI. You can also access the underlying Context Graph programmatically. See the section called “Using Context Graph”.

Find which services need your attention

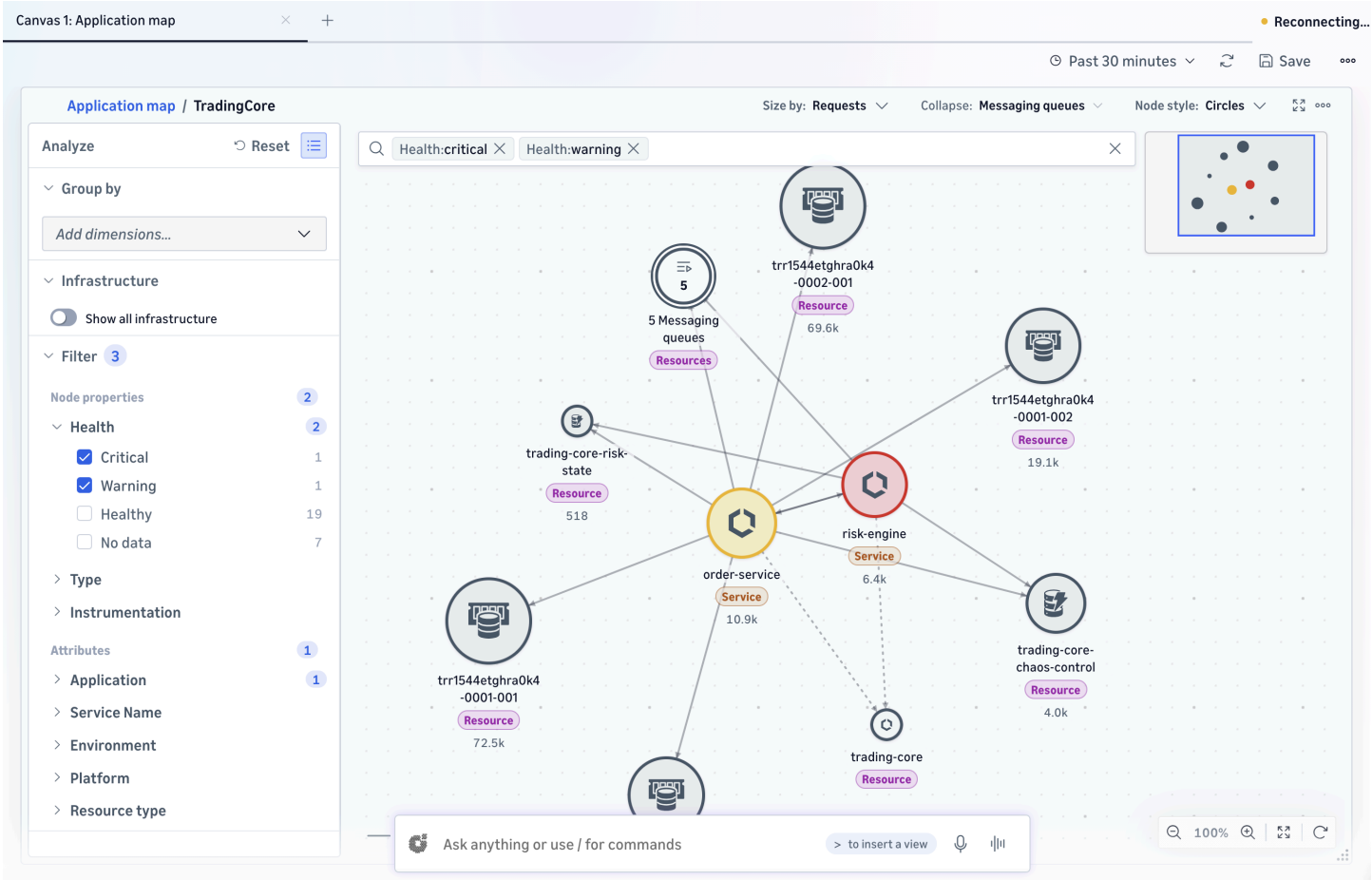
Start with the application map to find which applications have problems, and which service or dependency is responsible.

1. Select the time range that contains the behavior you want to investigate.
2. Find the application you want to review.
3. Use the **Health** filter to focus on services or dependencies that need attention. Health status for an application is "critical" if the percentage of errors is greater than 5%, and "warning" if it is greater than 1%.

4. Select a service to review its activity and relationships.
5. Open the service or its telemetry when you need more detailed evidence.

For detailed service monitoring, see the section called “Monitor a service”. To investigate individual requests and operations, see the section called “Application traces”.

Understand what appears on the map



The map is your application's observed topology, drawn from two kinds of nodes and three kinds of relationship between them.

Node	What it is
Service	An instrumented application that sends its own telemetry.

Node	What it is
Resource	A resource that a service uses or runs on, including managed and customer-deployed resources such as a database, queue, bucket, function, or compute.

Nodes also carry a category, which determines the icon shown on the map: compute, storage, database, messaging/queue, Gen AI model, network, or uncategorized.

Relationship	What it means
Calls	One service calls another service.
Accesses	A service reads from or writes to a resource.
Runs on	A service is hosted on a piece of infrastructure, such as a cluster.

A relationship can also contain operations observed between its nodes. For example, you can see which operations one service called on another service or resource. Node health reflects the error rate observed over the selected time range. A node is reported as one of the following:

- **Healthy** — the observed error rate is within normal range.
- **Warning** — the error rate is elevated.
- **Critical** — the error rate is high.
- **Unknown** — the node has too little traffic to classify. A low-traffic node is reported as unknown rather than as healthy.

Investigate an application or a service from the map

Select an application or a service to review information for the selected time range. The service view can include request activity, latency, errors, and its relationships with other services and resources. Review the services that call the selected service, the services it calls, and the resources

it accesses. This helps you determine whether the problem is isolated to the service or involves a dependency. From the service, continue to the telemetry:

1. Open traces to investigate individual requests.
2. Open logs to inspect application messages, exceptions, or events.
3. Open metrics to investigate service or infrastructure signals.

Narrow and organize the map

The map opens grouped by **Application**, so it shows your applications rather than every individual service. Use the **Analysis** rail when a large map contains more nodes than you need for the current investigation. Use **Group by** in the analysis rail to change the grouping dimension. For example, if your services run in multiple environments, group by **Environment** to focus on all services in a given environment.

Use **Filter** to narrow the map without changing its grouping. You can filter by node properties such as Health, Type, and Instrumentation, or by attributes such as **Application**, **Service Name**, **Environment**, **Platform**, and **Resource type**.

Use **Show all infrastructure** to include resources discovered in your account even when Omni has not observed them through telemetry. Use **Node style** to switch the node view between cards and circles.

What to watch out for

1. **Most relationships on the map come from your telemetry.** A service that is not instrumented appears as a single node with no related telemetry if Omni discovers it in your account. It can also appear as a node named from a caller's span, because the caller's telemetry names the far end of the call. The map draws a call between two of your services when you trace it. Omni also builds relationships from network and access logs, from CloudTrail, and from the resources it discovers in your account. To close a gap, instrument the service that is missing. See the section called “Send telemetry to CloudWatch Omni”.
2. **Individual hosts, tasks, and identities are left out on purpose.** The map does not create nodes for EC2 instances, ECS tasks and containers, or IAM roles and users. Leaving these out keeps the map focused on your services and their dependencies.
3. **Two nodes can carry the same service name.** Omni creates a separate node for each combination of AWS account, AWS Region, cloud provider, and deployment environment.

Each combination is a separate deployment of the service. The service details page shows a node's account, Region, and cloud provider as node attributes, so you can tell two deployments apart there. Set those attributes consistently. If some replicas of one service set `deployment.environment` and others leave it unset, Omni splits that service into more than one node and divides its traffic and dependencies between them. See the section called “Using Context Graph”.

4. **Resources that Omni finds by inspecting your account appear on a delay, and the time range does not filter them.** Telemetry updates the map continuously, and telemetry-derived nodes are the ones observed inside the range you select. A resource that Omni learns about by discovering what is deployed in your account appears after the next scheduled refresh. It then reflects the most recent discovery rather than the selected range, so narrowing the range does not remove it. Such a resource can appear with its name before anything has called it, and it shows no metrics or traces because Omni has not observed any. For which discovery sources contribute what, see the section called “Using Context Graph”.

Where to go next

To do this	See
Open a single service, its dependencies, and its health	the section called “Monitor a service”
Follow one request through your services	the section called “Application traces”
Query the logs, metrics, and traces behind a node	the section called “Explore and query your telemetry”
Understand spaces and where this telemetry is stored	*CloudWatch Omni*

Cross-account and cross-Region topology

The map shows the services and resources in the space's own account and Region. To add more accounts and Regions, centralize their telemetry into the space's account. Their services and dependencies then appear on the same map. To set this up, see the section called “Set up Omni for your organization”.

Using Context Graph

Use the Context Graph to understand the services, resources, and relationships that Omni has identified in your environment. Omni builds the graph from your telemetry and from resources it discovers in your account. A dependency is one kind of relationship: evidence that one component called another. The application map is a visual view of this graph. For the node and relationship types shown on the map, see the section called “Monitor applications with the application map”.

> **Note**
>
> Omni builds the graph from telemetry you are already sending and from the resources it finds in your account. You do not need to create or maintain the graph. The telemetry and resource attributes you send determine how completely and accurately Omni represents your environment.

Understand what contributes to the graph

Different telemetry signals contribute different information.

What you send	What it contributes
Traces	The caller, the callee, and the dependency between them. Traces establish service-to-service call relationships.
Metrics	A node for the component that emitted the metrics. Metrics alone do not establish a dependency.
Logs	A node for the component that emitted the logs. When a log identifies its compute, Omni can also associate the service with that compute.
Network and access logs	Communication observed between endpoints . Omni can resolve those endpoints to known

What you send	What it contributes
	resources when enough resource information is available.

A dependency appears when Omni has evidence that one component called another. A trace is the most direct evidence. Network logs, access logs, and CloudTrail also carry that evidence for some call paths. Metrics do not, so two services that only emit metrics can both appear with no dependency between them. To capture service-to-service dependencies reliably, instrument the caller and send traces. See the section called “Send telemetry to CloudWatch Omni”.

Understand discovered resources

Omni can also add resources that it discovers in your account. A discovered resource can appear in the graph before Omni observes telemetry from that resource. In that case, the resource exists in the graph but has no telemetry attached to it. This is expected and does not necessarily indicate an instrumentation problem.

Telemetry and discovery use different time semantics. Discovered resources reflect the most recent inventory available to Omni. Changing the telemetry time range does not remove a discovered resource that had no activity during that period. A resource can therefore exist in the graph without any observations in the selected time range.

Discovery source	What it adds
AWS Config	Resources that AWS Config records in your account. This source adds resource nodes. It also adds some relationships, such as the compute a service runs on.
Resource discovery	Services and resources found by inspecting your account, and the relationships between them: which compute a service runs on, and which resources it accesses.

Omni creates a service-linked configuration recorder when you create a space. AWS Config then records a fixed set of resource types. Omni adds those resources to the Context Graph. You do not

create the recorder. You do not configure it. You pay no AWS Config fee for the records that Omni uses.

Omni adds a graph node for only some of the types. It uses the other records to add detail to nodes that exist. For the full list, see the section called “Using AWS Config for resource discovery”.

> **Important**
>
> **Discovery reports current state rather than a time window.** Telemetry-derived elements are the ones observed inside the time range you request. Discovery-derived elements reflect the most recent discovery of your account, so narrowing the time range does not narrow them. A graph made only of discovered resources reports no observations in the range at all.

Make services appear correctly

Use consistent OpenTelemetry resource attributes so Omni can identify services and deployments correctly.

Identify each service: Set `service.name` on every service. If a service is missing from the graph, verify this attribute first.

Group services into an application: Set `service.namespace` on related services so they are grouped into the same application.

Distinguish deployments: Use `deployment.environment` to distinguish deployments such as production and staging.

> **Important**
>
> Set identity attributes consistently across every replica of a service. Two replicas that declare **different** values become separate nodes and split their telemetry. A replica that leaves an attribute unset folds onto the replica that declares it.

Identify managed dependencies: Add the attributes that name the destination to the calling span. Omni then resolves the dependency to a known resource. OpenTelemetry semantic conventions define these attributes. Supported AWS SDK instrumentation can set many of them for you.

When the call is to	The span needs
A queue or topic	`messaging.destination.name`
A database or table	`db.namespace` (or `db.name`). Add `db.collection.name` for the table or collection.
An object store bucket	`aws.s3.bucket`
A function	`faas.invoked_name`
A model	`gen_ai.request.model`

If a span contains only a hostname, Omni can represent the dependency by that hostname rather than by the underlying resource. For a call between two services, the span must also represent the outbound call: a span that records only work received does not establish an outgoing dependency to another service. Messaging is the exception. A consumer that polls a queue or a topic gets a relationship to that queue or topic.

Retrieve the graph programmatically

Use the `GetContextGraph` operation to retrieve the Context Graph through the API. Authenticate the request the same way as other Omni API operations.

Each request includes a time range. Nodes and relationships from telemetry show the activity in that range. Discovered resources show the latest inventory instead. The time range does not limit them. Results can span several pages.

> **Important**
>
> Relationships are returned with their source nodes rather than as one top-level edge list. When you process a response, iterate through the nodes and read the relationships attached to each node.

Omni attaches each relationship to the node it starts from. A node carries all of its outbound relationships on the page that carries the node. They are never split across pages. To find what

points at a node, you must read the nodes at the far end, and those nodes can be on a later page. Read all pages before you decide what depends on a node.

Node and relationship kinds in the response use the same model the application map presents. Each node also reports which signals it was seen in and which sources contributed it. The graph carries no evidence that a dependency does *not* exist.

Pivot from the graph to your telemetry

Each node carries the attribute values that identify its own telemetry, so you can move from a dependency on the graph to the metrics, logs, and traces behind it. Use those attribute values as the selectors in your query. For the query languages, see the section called “Explore and query your telemetry”.

This applies to nodes that telemetry contributed. A resource that Omni discovered in your account and never saw in telemetry carries no signals, no operations, and no metrics, so check `signalTypes` before you query a node's telemetry. An empty `signalTypes` means Omni observed no telemetry for that node inside the range you requested. It does not mean the resource was never used.

A node reports its attributes in several forms, and the difference matters when you query.

On the node	What it is, and when to use it
Normalized identity properties	The resolved identity Omni settled on after merging what it observed across sources. Use these to identify and display a node.
Telemetry attributes	The OpenTelemetry attributes as your telemetry emitted them. Use these to query, because they are the values the stored telemetry carries.
Operation details	For each operation observed on the node, the attribute sets that identify that operation's metric series. Use these attribute sets to chart one operation rather than the whole service.
Per-metric metadata	For each metric on the node: its name, its meaning, its unit, the statistic to chart, and

On the node	What it is, and when to use it
	the attributes to query it with. Request this explicitly. See the following note.

> **Note**
>
> Omni returns per-metric metadata only when you ask for it, because it costs an extra lookup for every node in the response. Operation details are returned either way.

> **Important**
>
> **Query a metric using the attributes reported for that metric rather than the node's identity.** A node that Omni merged from several sources can carry different attribute values for different metrics. The per-metric attributes match the emitted series. The node's resolved identity might not match, and a query built from it can return nothing even though the metric exists.

A node that telemetry contributed also reports when Omni first and most recently observed it. Compare those timestamps, or run the same request over two ranges, to see whether something appeared, disappeared, or stopped being exercised. A resource known only from discovery might report neither, because Omni has not observed it in use.

Name your operations

Omni names an operation from the semantic-convention attributes on the span, and it prefers the most specific set of attributes. Omni also records the span name as an operation. When the two names match, they merge into one operation. When they differ, you see both. If a span records the route it served, Omni names the operation for that route rather than for the method alone.

Record this	And an operation is named
`http.request.method` and `http.route`	By method and route together, such as a GET on a specific path, rather than by method alone

Record this	And an operation is named
`db.operation.name` with `db.namespace` or the table	By the database operation and what it acted on
`messaging.destination.name` and `messaging.operation.type`	By the destination and the kind of messaging operation

Record the route. Without it, Omni names one operation for each HTTP method, and every request the service handles is grouped into that operation.

Use the graph with an AI agent

An AI agent can use the Context Graph to retrieve topology while answering a question or investigating a problem. The agent can verify relationships between services and resources before using those relationships as evidence in its analysis.

For how to connect an agent to Omni, see the section called “Use Omni skills”.

Troubleshoot an incomplete graph

Check the following when the graph does not show what you expect:

1. **Relationships come from your telemetry.** A service-to-service dependency requires trace evidence of the call, so a call that is never traced is not a relationship. Other signals identify a component without establishing the dependency. A service that sends no telemetry appears only if Omni discovers it in your account, and it then appears with no telemetry attached.
2. **A single page is not the whole graph.** One response page can contain only part of the graph, or only some of a node's relationships.
3. **Omni identifies a resource from its resource identifier.** Where Omni can resolve a managed resource to a resource identifier, it represents the resource. Where it cannot, the same dependency can appear as a host, an endpoint, or an address instead.
4. **The time range changes what telemetry contributes.** Telemetry-derived nodes and relationships appear only when Omni observed them during the range you request, so a dependency exercised once a day is absent from a one-hour window. That absence is not evidence that the dependency does not exist.

5. **Discovered resources report current inventory.** Resource discovery is not constrained by the telemetry time range, so discovered resources reflect the latest inventory regardless of the range you choose.

Where to go next

To do this	See
Read the map, group it, and open a service from it	the section called “Monitor applications with the application map”
Instrument a service so it appears in the graph	the section called “Send telemetry to CloudWatch Omni”
Query the telemetry behind a node	the section called “Explore and query your telemetry”

Using AWS Config for resource discovery

CloudWatch Omni uses AWS Config to find the AWS resources in your account. Omni then adds those resources to the context graph. This page lists the resource types that AWS Config records for Omni, and describes how the recorder works.

To learn how discovered resources appear in the graph, see the section called “Using Context Graph”.

How the recorder works

Omni creates a *service-linked configuration recorder* when you create a space. The recorder is named `AWSConfigurationRecorderForCloudWatch`. You do not create it. You do not configure it.

A service-linked recorder belongs to the service that it is linked to. That service sets the resource types to record. To learn more, see Working with the configuration recorder in the *AWS Config Developer Guide*.

The recorder works as follows:

- **You pay no AWS Config fee.** Omni uses an internal recording scope. AWS Config does not send these records to your delivery channel, and you cannot read them in AWS Config. You can still see the recorder itself in the AWS Config console.
- **The recorder's settings are fixed.** You cannot change them in AWS Config. You cannot rename the recorder, and you cannot start it or stop it. Omni sets which resource types the recorder records, and it records the same types in every account.
- **You can delete the recorder, but not while a space uses it.** When you call `DeleteServiceLinkedConfigurationRecorder`, AWS Config asks Omni whether it still uses the recorder. While your space has an active AWS Config integration, Omni answers yes and your call fails. To remove the recorder, delete the space first.
- **Deleting the recorder stops resource discovery.** Omni keeps the resources that it already found, and it adds no new ones. AWS Config keeps the records that it already made.
- **Your own recorder is separate.** Omni does not change or use your customer managed recorder.

> **Note**
>
> Other CloudWatch features also create service-linked recorders. Your account can hold more than one recorder that is linked to CloudWatch.

Which resources become graph nodes

AWS Config records every type in the resource-type table on this page. Omni adds a graph node for only some of those types. Omni uses the other records to add detail to nodes that exist. For example, Omni reads subnet and security group records. It uses them to find the network address of a service.

A resource can appear in the graph before Omni gets telemetry from it. To learn more, see the section called “Using Context Graph”.

Which Regions are recorded

Omni creates the recorder in the same Region as your space. The recorder records the resources in that Region.

To add the resources from another Region, centralize that Region's telemetry into the space's account. For more information, see the section called “Set up Omni for your organization”.

AWS Config makes a record only for a type that AWS Config supports in the Region. If AWS Config does not support a type in a Region, no resource of that type appears in the graph for that Region.

Describe calls in CloudTrail

AWS Config takes an inventory of the resources in your account at regular times. It does this for all resources, not only for the types that this page lists. You can therefore see `ConfigResourceCompositionSession` and `AWSConfig-Describe` calls in CloudTrail, including calls for types that Omni does not record. This behavior keeps the AWS Config inventory consistent.

Recorded resource types

AWS Config records the following resource types for Omni.

Category	AWS service	Resource types
Compute	AWS App Runner	`AWS::AppRunner::Service`
Compute	AWS Elastic Beanstalk	`AWS::ElasticBeanstalk::Environment`
Compute	AWS Lambda	`AWS::Lambda::Function`
Compute	Amazon EC2	`AWS::EC2::Instance` `AWS::EC2::LaunchTemplate`
Compute	Amazon EC2 Auto Scaling	`AWS::AutoScaling::AutoScalingGroup` `AWS::AutoScaling::LaunchConfiguration`
Compute	Amazon ECS	`AWS::ECS::CapacityProvider` `AWS::ECS::Cluster` `AWS::ECS::Service` `AWS::ECS:`

Category	AWS service	Resource types
		:TaskDefinition AWS::ECS::TaskSet
Compute	Amazon EKS	AWS::EKS::Cluster AWS::EKS::FargateProfile AWS::EKS::Nodegroup
Compute	Amazon WorkSpaces	AWS::WorkSpaces::Workspace
Networking and content delivery	AWS Global Accelerator	AWS::GlobalAccelerator::Accelerator
Networking and content delivery	Amazon CloudFront	AWS::CloudFront::Distribution
Networking and content delivery	Amazon Route 53	AWS::Route53::HostedZone

Category	AWS service	Resource types
Networking and content delivery	Amazon VPC	AWS::EC2::ClientVpnEndpoint AWS::EC2::EIP AWS::EC2::FlowLog AWS::EC2::InternetGateway AWS::EC2::NatGateway AWS::EC2::NetworkInterface AWS::EC2::RouteTable AWS::EC2::SecurityGroup AWS::EC2::Subnet AWS::EC2::TransitGateway AWS::EC2::TransitGatewayAttachment AWS::EC2::VPC AWS::EC2::VPCEndpoint AWS::EC2::VPCPeeringConnection AWS::EC2::VPNConnection AWS::EC2::VPNGateway
Networking and content delivery	Elastic Load Balancing	AWS::ElasticLoadBalancing::LoadBalancer AWS::ElasticLoadBalancingV2::Listener AWS::ElasticLoadBalancingV2::LoadBalancer AWS::ElasticLoadBalancingV2::TargetGroup

Category	AWS service	Resource types
APIs	AWS AppSync	`AWS::AppSync::GraphQLApi`
APIs	Amazon API Gateway	`AWS::ApiGateway::Method AWS::ApiGateway::RestApi AWS::ApiGateway::Stage AWS::ApiGatewayV2::Api AWS::ApiGatewayV2::Integration AWS::ApiGatewayV2::Stage`
Databases and storage	Amazon DynamoDB	`AWS::DynamoDB::Table`
Databases and storage	Amazon EFS	`AWS::EFS::FileSystem`
Databases and storage	Amazon Keyspaces	`AWS::Cassandra::Keyspace`
Databases and storage	Amazon OpenSearch Service	`AWS::Elasticsearch::Domain AWS::OpenSearch::Domain`
Databases and storage	Amazon QLDB	`AWS::QLDB::Ledger`
Databases and storage	Amazon RDS	`AWS::RDS::DBCluster AWS::RDS::DBInstance`
Databases and storage	Amazon Redshift	`AWS::Redshift::Cluster`
Databases and storage	Amazon S3	`AWS::S3::AccessPoint AWS::S3::Bucket`
Application integration	AWS IoT Core	`AWS::IoT::TopicRule`

Category	AWS service	Resource types
Application integration	AWS Step Functions	AWS::StepFunctions::Activity AWS::StepFunctions::StateMachine
Application integration	Amazon Data Firehose	AWS::KinesisFirehose::DeliveryStream
Application integration	Amazon EventBridge	AWS::Events::ApiDestination AWS::Events::Connection AWS::Events::EventBus AWS::Events::Rule
Application integration	Amazon EventBridge Scheduler	AWS::Scheduler::ScheduleGroup
Application integration	Amazon Kinesis Data Streams	AWS::Kinesis::Stream
Application integration	Amazon MQ	AWS::AmazonMQ::Broker
Application integration	Amazon MSK	AWS::MSK::Cluster AWS::MSK::ServerlessCluster
Application integration	Amazon SNS	AWS::SNS::Topic
Application integration	Amazon SQS	AWS::SQS::Queue

Category	AWS service	Resource types
Containers and service discovery	AWS Cloud Map	`AWS::ServiceDiscovery::HttpNamespace AWS::ServiceDiscovery::PublicDnsNamespace AWS::ServiceDiscovery::Service`
Containers and service discovery	Amazon ECR	`AWS::ECR::PublicRepository AWS::ECR::Repository`
Security and identity	AWS Certificate Manager	`AWS::ACM::Certificate`
Security and identity	AWS IAM	`AWS::IAM::Policy AWS::IAM::Role`
Security and identity	AWS KMS	`AWS::KMS::Key`
Security and identity	AWS Network Firewall	`AWS::NetworkFirewall::Firewall`
Security and identity	AWS Secrets Manager	`AWS::SecretsManager::Secret`
Security and identity	AWS WAF	`AWS::WAFv2::WebACL`
Security and identity	Amazon Cognito	`AWS::Cognito::IdentityPool AWS::Cognito::UserPool`
Management and governance	AWS CloudFormation	`AWS::CloudFormation::Stack`
Management and governance	AWS CloudTrail	`AWS::CloudTrail::Trail`

Category	AWS service	Resource types
Management and governance	Amazon CloudWatch	AWS::CloudWatch::Alarm AWS::CloudWatch::MetricStream
Management and governance	Amazon CloudWatch Logs	AWS::Logs::Destination
Machine learning	Amazon Bedrock	AWS::Bedrock::DataSource AWS::Bedrock::Guardrail AWS::Bedrock::KnowledgeBase AWS::Bedrock::Prompt
Machine learning	Amazon Kendra	AWS::Kendra::Index
Machine learning	Amazon Lex	AWS::Lex::Bot AWS::Lex::BotAlias
Machine learning	Amazon SageMaker AI	AWS::SageMaker::Domain AWS::SageMaker::EndpointConfig AWS::SageMaker::Model AWS::SageMaker::NotebookInstance
Analytics	AWS Glue	AWS::Glue::Crawler AWS::Glue::Database AWS::Glue::Job
Analytics	Amazon Athena	AWS::Athena::WorkGroup

Category	AWS service	Resource types
Analytics	Amazon QuickSight	`AWS::QuickSight::DataSource`
Developer tools and migration	AWS CodeBuild	`AWS::CodeBuild::Project`
Developer tools and migration	AWS CodeDeploy	`AWS::CodeDeploy::Application` `AWS::CodeDeploy::DeploymentGroup`
Developer tools and migration	AWS CodePipeline	`AWS::CodePipeline::Pipeline`
Developer tools and migration	AWS DMS	`AWS::DMS::ReplicationInstance`
Developer tools and migration	AWS Transfer Family	`AWS::Transfer::Server`
Developer tools and migration	Amazon AppFlow	`AWS::AppFlow::Flow`

Related information

- the section called “Using Context Graph”
- the section called “Monitor applications with the application map”
- Working with the configuration recorder in the *AWS Config Developer Guide*

Monitor a service

Open a service

Open a service from the application map or services list. The service view shows information for the selected time range.

Use the service signals to narrow the problem

Start in the service details view to decide what kind of issue you are seeing. Review the following telemetry:

- **Service properties**, such as the service name and identifying attributes for application, environment, cloud provider, Region, or account.
- **Requests** to spot traffic changes.
- **Request Latency** to identify slowdowns affecting the service.
- **Errors** to see whether requests are failing.
- **Trace summary** to see how much trace data exists for the service and how much of it includes errors.

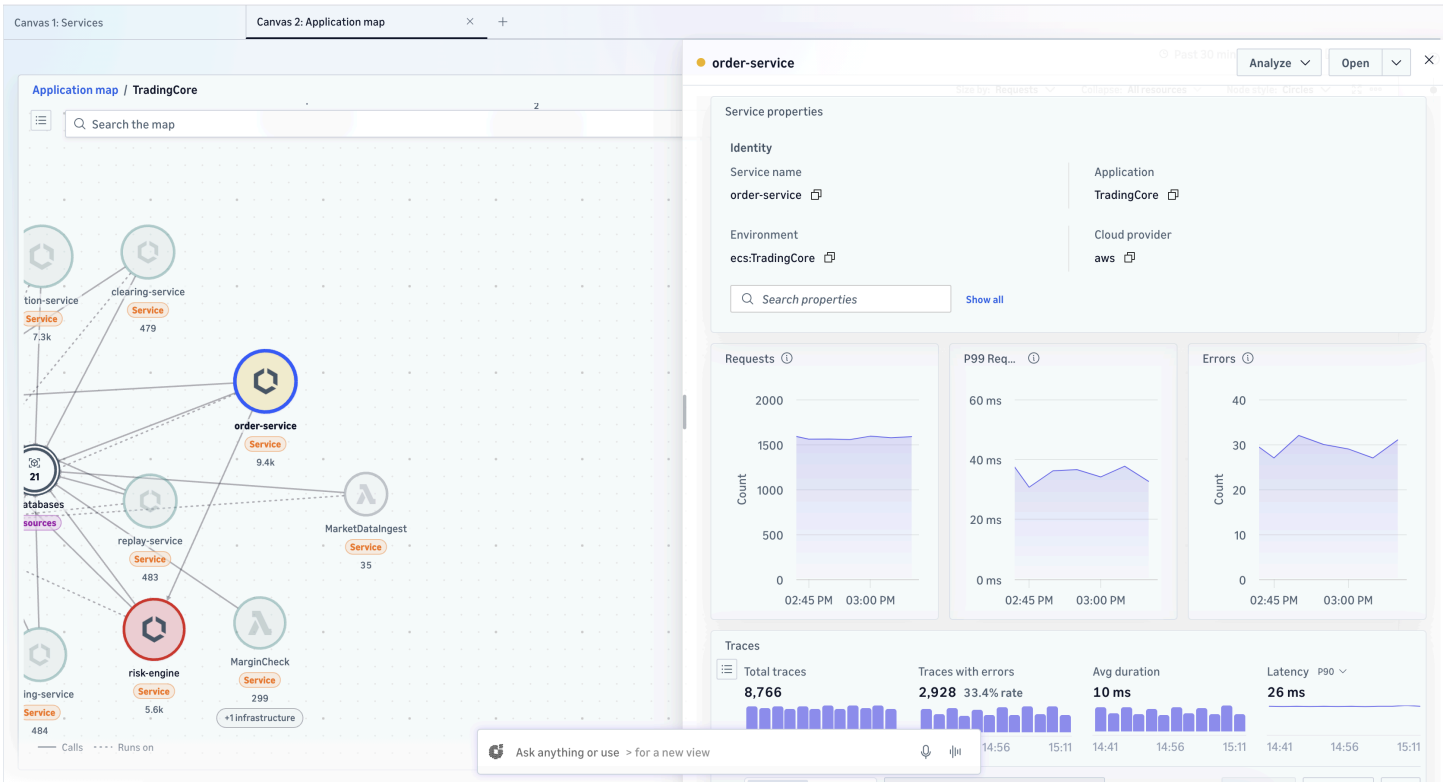
Together, these signals help you decide whether to investigate traces, logs, or broader service behavior.

Continue the investigation from the service view

From the service details view, you can continue directly into the telemetry behind the service. Use the following actions:

- From a service, choose **Analyze traces** or **View traces**.
- From a point on a metric chart, choose **Search for traces** to see the traces from exactly that time bucket or **Open in Explore** to continue analysis in Explore. On a latency chart the option is **Search for slow traces**, biased toward the slow requests behind the point you chose.

For how to read the traces list, inspect a trace's spans, and pivot from a span to logs or service metrics, see the section called “Application traces”. For an agent's own runs, see the section called “Agent traces”. To query trace data directly instead, see the section called “Explore and query your telemetry”.



What to watch out for

- **A trace ends where instrumentation ends.** A call to an uninstrumented service shows as the caller's outgoing span with nothing deeper. To extend the trace, instrument the callee. See the section called “Send telemetry to CloudWatch Omni”.
- **Errors can read Not available.** The **Errors** metric needs a dimension that not all telemetry carries; when it is absent, the service view says **Not available** rather than showing a misleading zero. Faults are unaffected.

Populate the request metrics with span metrics

The request metrics (**Requests**, **Errors**, and **Latency**) are derived from span metrics. To emit span metrics, add the CloudWatch plugin for OpenTelemetry to your application's instrumentation. For details, see the section called “Send telemetry to CloudWatch Omni” and the section called “Send application telemetry”.

Where to go next

To do this	See
Query trace data with SQL, or pivot by trace ID	the section called “Explore and query your telemetry”
Explore an agent's own traces and sessions	the section called “Agent traces” and the section called “Agent sessions”
Investigate an individual request end to end	the section called “Application traces”
See the whole system a service belongs to	the section called “Monitor applications with the application map”

Application traces

An application trace records one request as it passes through your services and dependencies. A span records one operation within that trace. Use this page to find a request, inspect the operations inside it, and open the related logs or service metrics.

For traces of AI agent runs, see the section called “Agent traces”.

Available in the Omni web UI only.

Open application traces

Choose the entry point that matches what you are investigating:

- From the launcher, under Application performance monitoring, choose `Traces` to explore requests directly.
- From the application map, select a service and choose **Analyze traces**.
- From Services, open a service and choose **Analyze traces**.

Select the time range you want to investigate.

Trace context does not propagate by itself: both sides of each hop must be instrumented. If the next service in a request is not instrumented, the trace stops there.

Choose traces or spans

Choose the investigation level that matches your question.

Traces

Choose **Traces** when you want to investigate a complete request.

Use traces for questions such as:

- Which requests failed?
- Which requests were slow?
- Which trace should you inspect in detail?

The summary above the results shows information such as **Total traces** and **Traces with errors** for the selected time range.

Spans

Choose **Spans** when you want to search individual operations across traces.

Use spans for questions such as:

- Which `DynamoDB.PutItem` operations reported errors?
- Which dependency operations took the longest?
- Which operations ran in a particular service?
- Which spans match a status or attribute?

Use **Search spans** to find an operation by name. Use the status control or **Errors** filter to narrow the results.

The results show when each span occurred and the operation it represents.

Investigate a request

1. Select the relevant time range.
2. Use **Traces** to find a complete request, or use **Spans** to find a specific operation.
3. Select a trace to review the operations that occurred during the request.
4. Find the span where an error occurred or where the request spent unexpected time.

5. Open related logs or service metrics when you need more evidence.

For aggregate application topology, see the section called “Monitor applications with the application map”. For one service's health and telemetry, see the section called “Monitor a service”.

Inspect a trace

Select a trace to open its detail view. The trace header shows the trace ID and the number of spans in the trace.

Switch among the available views:

- Timeline
- Graph
- Flame
- JSON

Use the Timeline view to follow span timing and parent-child relationships.

You can also:

- Search for a span within the trace.
- Filter spans by status.
- Choose Tree to follow parent-child structure.
- Choose Grouped to organize similar spans together.
- Use Errors to focus on spans that reported an error.
- Choose Reset to remove the current trace-detail filters.

Select a span to inspect its details.

Inspect trace or span details

Use the Trace and Span controls in the detail view to change the scope of the information shown. Choose Trace to review information about the complete request. Choose Span to review the selected operation.

The span detail can include:

- Name
- Span ID
- Parent ID
- Trace ID
- Service
- Start
- End
- Scope
- Status and duration

Expand the following sections when you need more detail:

- Span attributes
- Resource attributes
- Annotations
- Raw data

View logs for a trace

Logs can provide details that are not present in the trace, such as application messages, exceptions, and business events. From the trace header, choose **Show logs for this trace**. The logs view opens the records associated with the trace when the required correlation information is available.

If no logs appear, confirm that:

- The application produced logs during the request.
- Those logs were sent to your space.
- The selected time range includes the request.
- The log records contain the trace ID used for correlation.

Where supported by your instrumentation, include span and service context in log records as well, so you can narrow logs to the operation and service under investigation. For telemetry configuration, see the section called “Send telemetry to CloudWatch Omni”.

View metrics for a service

Choose **Show metrics for this service** to investigate the metrics associated with the service in the current trace context.

Use service metrics to determine whether the problem in the trace occurred alongside a broader change in request volume, latency, errors, or infrastructure behavior.

For service-level monitoring, see the section called “Monitor a service”.

When trace information is incomplete

Trace investigation depends on the telemetry your applications send. Use the following checks when information is missing:

- No traces appear. Confirm the time range and verify that the workload is sending traces.
- A trace stops at one service. Confirm that the next service is instrumented and propagates trace context.
- A trace has only one or a few flat spans. Verify that the workload records the internal operations you need to inspect.
- Related logs do not appear. Confirm log ingestion and trace-ID correlation.
- Service metrics do not appear. Confirm that the service publishes the required metrics.

Where to go next

To do this	See or choose
Review the service involved in the trace	the section called “Monitor a service”
See application topology and dependencies	the section called “Monitor applications with the application map”
Open logs associated with the request	Show logs for this trace
Examine metrics for the service	Show metrics for this service
Query trace or log records directly	the section called “Explore and query your telemetry”

Investigate an AI agent run instead	the section called “Agent traces”

Connect Slack

An integration connects CloudWatch Omni to a tool your team already uses. Slack delivers alert notifications to your team's channels, and lets your team mention the Omni agent there. You manage it in **Settings**, on the **Integrations** page, and a connection applies to your whole space.

With Slack connected, an alert's notification rules can deliver to Slack channels. You choose the channels on each alert when you set its destinations. See the section called “Alerts”.

Connecting the integration is not enough: delivery also requires the alert's access profile to grant this integration and the channels the alert posts to. Omni checks the grant when you create the alert and again on every delivery, so revoking it stops notifications for alerts that were already working. If an alert fires but nothing reaches Slack, check the profile. See the section called “Access profiles”.

To connect:

1. In **Settings**, choose **Integrations**.
2. On the **Slack** card, choose **Enable**. Slack's authorization page opens in a new tab.
3. Review the requested permissions and choose **Allow**.

Back in Omni, the **Slack** card shows the connected integration: its ID and name. If your Slack workspace requires administrator approval for new apps, ask a workspace administrator to approve the request, then connect again.

What the Slack connection can do. The authorization page lists the exact permissions requested. The connection lets Omni post and reply to messages, read the names of your workspace's public and bot-added private channels so it can offer them as destinations, and receive messages that mention the Omni app. It does not include your workspace's message history. A private channel appears in the channel list only after the Omni app has been added to it.

Bring the Omni agent into Slack. With the integration connected, your team can mention the Omni agent in a Slack channel and get answers there. Connecting the workspace is not enough: the `DefaultAccessProfileForAsyncWorkflows` access profile must also grant the agent permission to post to the integration. Without that grant, a mention is received but the agent

cannot respond. For what the agent can do, see the section called “Ask the Omni agent”. For profiles and grants, see the section called “Access profiles”.

> **Important**
>
> Amazon CloudWatch uses generative AI to produce the agent's replies. A reply can be incomplete or wrong. Verify a finding before you act on it. AWS handles information in accordance with the AWS Privacy Notice.

On the connected **Slack** card, choose **Set up Omni permissions for Slack**. This opens the add-grant form already filled in for the `DefaultAccessProfileForAsyncWorkflows` profile, where you choose the channels the agent may post to. Give the profile a viewer grant as well, so the agent can read the resources it answers about.

The grant covers three things:

- **Reply through the Slack integration** — `cloudwatch:InvokeIntegration`, scoped to the Slack integration's ARN and the channels you choose.
- **Work with conversation threads** — `cloudwatch:GetOmniThread` and `cloudwatch:CreateOmniThread`. A viewer grant already includes both, so a profile that has one does not need them listed separately.
- **Read the resources it answers about** — a viewer grant.

If you author grants through the API instead of the console, use your own integration ARN and channel names:

```
{
  "name": "slack-agent-integration",
  "permission": "CUSTOM",
  "scopedActions": [
    {
      "actions": ["cloudwatch:InvokeIntegration"],
      "resources": [
        {
          "resourceType": "Integration",
          "resourceArns": ["arn:aws:cloudwatch:region:account-id:integration/
integration-id"]
        }
```

```
      ],
      "contextConditions": { "channelNames": ["#oncall-alerts", "#incidents"] }
    },
    { "actions": ["cloudwatch:GetOmniThread", "cloudwatch:CreateOmniThread"] }
  ]
}
```

Pair it with a viewer grant:

```
{ "name": "slack-agent-viewer", "permission": "READ" }
```

Regional Slack apps. The Slack app is Region-specific, and you do not choose it: **Enable** directs you to the Amazon CloudWatch Slack app for the Region your space is in. It appears in your workspace as a bot with a matching name. Within one Region, a Slack workspace can be connected to only one space. The same workspace can be connected to spaces in different Regions. Each Region installs its own bot, so mention the bot whose name matches the Region of the space you want to reach.

Invite the bot to each channel. Connecting the workspace does not add the bot to any channel. Before an alert can deliver notifications to a channel, or a mention in that channel can reach the agent, invite the Region's bot to it from Slack:

```
/invite @Amazon CloudWatch - USEast Virginia
```

Region	Bot name
US East (N. Virginia), `us-east-1`	`Amazon CloudWatch - USEast Virginia`
US East (Ohio), `us-east-2`	`Amazon CloudWatch - US East (Ohio)`
US West (Oregon), `us-west-2`	`Amazon CloudWatch - US West Oregon`
Europe (Ireland), `eu-west-1`	`Amazon CloudWatch - EU (Ireland)`
Asia Pacific (Mumbai), `ap-south-1`	`Amazon CloudWatch - AP (Mumbai)`
Asia Pacific (Singapore), `ap-southeast-1`	`Amazon CloudWatch - AP (Singapore)`

Region	Bot name
Asia Pacific (Sydney), `ap-southeast-2`	`Amazon CloudWatch - AP (Sydney)`
Asia Pacific (Tokyo), `ap-northeast-1`	`Amazon CloudWatch - AP (Tokyo)`
Canada (Central), `ca-central-1`	`Amazon CloudWatch - CA (Central)`
Europe (Frankfurt), `eu-central-1`	`Amazon CloudWatch - EU (Frankfurt)`
Europe (London), `eu-west-2`	`Amazon CloudWatch - EU (London)`
South America (São Paulo), `sa-east-1`	`Amazon CloudWatch - SA (Sao Paulo)`

Disconnect an integration

On the integration's card in **Settings**, on the **Integrations** page, choose **Disable**. Disconnecting Slack stops alert notifications to Slack. The alerts themselves keep evaluating, and their other destinations keep working. You can reconnect with **Enable**.

Who can manage integrations

Integrations apply to the whole space. One person connects, and the connection works for everyone in the space. Whether a member can connect, change, or disconnect an integration is governed by their grants: the integration create, update, and delete actions. See the section called "Custom grant actions".

Use Omni skills

You can use CloudWatch Omni from the AI coding agent you already develop with — including Claude Code, Cursor, Kiro, and Codex — by installing the Omni skills from the Agent Toolkit for AWS. A skill is a set of task-focused instructions, reference material, and scripts that your coding agent loads when a task matches, so it can complete Omni tasks: set up instrumentation, query your telemetry, or work with alerts and dashboards.

> **Note**
>
> This page is about using Omni from your own coding agent. For the AI assistance built into the IDE extension, see the section called "Develop with the IDE extension". For the AI assistant in the Omni web UI, see the section called "Ask the Omni agent".

How skills work with your AWS credentials

When you give your coding agent an Omni task, the agent activates the matching skill and follows its instructions. The scripts included with the skills call the public AWS API endpoints that power Omni (Amazon CloudWatch, Amazon Bedrock AgentCore, and Amazon Bedrock) using the AWS credentials configured in your environment, for example the credentials you get from running `aws sso login`. This is the same authentication the public API uses.

What your coding agent can do with Omni

With the Omni skills installed, your coding agent can help you:

- Set up Omni observability for your application or AI agent, including instrumentation.
- Query the telemetry in your CloudWatch Dataset and work with the results.
- Manage Omni resources such as alerts, dashboards, and agent evaluations.
- Build datasets from traces for agent evaluation.
- Connect external integrations such as Slack.

Install the Omni skills

Before you begin, you need:

- An AWS account with Omni set up. See the section called "Set up Omni".
- AWS credentials configured in your development environment.
- A coding agent that supports skills, such as Claude Code, Cursor, Kiro, or Codex.

To install and use the Omni skills

1. Install the skills from the Getting started page in the Agent Toolkit for AWS User Guide, which has installation instructions for each supported coding agent.
2. Give your coding agent an Omni task, such as "Set up Omni observability for this application" or "Instrument this agent and send its traces to Omni."
3. Verify the result in the Omni web UI. For example, confirm that traces arrive or that the alert or dashboard the agent created matches what you asked for.

Security in CloudWatch Omni

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. For the compliance programs that apply to CloudWatch Omni, see the section called “Compliance validation for CloudWatch Omni”.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when you use CloudWatch Omni. For a space, your share covers who can read your telemetry and which key encrypts what CloudWatch Omni stores. The following topics show you how to configure both, and how to use other AWS services that help you secure a space. To control what content reaches your telemetry before CloudWatch Omni stores it, see the section called “Protect sensitive data”.

Topics

- the section called “Data encryption in CloudWatch Omni”
- the section called “Identity and access management for CloudWatch Omni”

- the section called “Compliance validation for CloudWatch Omni”
- the section called “Resilience in CloudWatch Omni”
- the section called “Infrastructure security in CloudWatch Omni”
- the section called “Configuration and vulnerability analysis in CloudWatch Omni”
- the section called “Cross-service confused deputy prevention”
- the section called “Security best practices for CloudWatch Omni”
- the section called “Opting out of using your data for service improvement”
- the section called “IAM policies to use CloudWatch Omni”
- the section called “Cross-Region inference in CloudWatch Omni”

Data encryption in CloudWatch Omni

CloudWatch Omni encrypts your data at rest and in transit. This page describes what is encrypted, which keys are used, and what your AWS KMS key policy must allow.

To choose what content enters your telemetry, see the section called “Protect sensitive data”. To configure a customer managed key on your space, see the section called “Set up Omni”.

Encryption at rest

Store	Default encryption	Customer managed key
Telemetry in your space, held in the CloudWatch Dataset	Managed by CloudWatch Omni	Supported for enabled accounts. Configure the key on the space.
Evaluation job and configuration metadata	AWS KMS key	Supported
Staging storage in Amazon S3	SSE-S3, with versioning and TLS-enforced access	—
Forward access session credentials	A dedicated service-managed AWS KMS key. These credentials are envelope-encrypted, bound by	Service-managed

Store	Default encryption	Customer managed key
	encryption context to the job that uses them, and never written to logs.	

> **Important**
>
> Customer managed key support is enabled per account, so the option might not appear when you configure your space. Removing a customer managed key from a space requires `kms:Decrypt` on the outgoing key, so the caller can still read data already written under it.

Files that CloudWatch Omni writes to your own machine are not encrypted by the service. When you use the IDE extension and local development tools, telemetry can be written under a local `.omni/` directory, including raw prompts and responses. These files are plaintext and rely on your operating-system file permissions. Local mode keeps your telemetry off the network until you choose to send it, but it does not encrypt it at rest. Protecting these files is your responsibility, and the sample `.gitignore` that ships with the tooling re-includes several `.omni/` data files, so raw trace content can be committed to source control. Available in the IDE extension only.

Key management

If you use a customer managed AWS KMS key, your key policy must allow the following. Your policy needs **both** of the first two statements: when you attach the key with AssociateDatasetKmsKey, CloudWatch checks your own access to the key before it checks the service's, so a policy that grants only the CloudWatch service principal is denied for the caller and the key is rejected.

1. **Dry-run checks signed by the CloudWatch service principal.** When you attach a customer managed key to a space, CloudWatch Omni performs dry-run `kms:Decrypt` and `kms:GenerateDataKey` calls against your key, using the encryption context `{"aws:cloudwatch:arn":"arn:aws:cloudwatch:<region>:<account-id>:space/<space-id>"}`. Both must succeed, or the key is rejected when you attach it. The service repeats the check on later requests. A key policy that is narrowed after you attach the key fails closed rather than silently. These calls are signed by the CloudWatch service principal, `cloudwatch.amazonaws.com`, and not by you. Grant that principal both actions on the encryption context above, and condition the statement on

`aws:SourceArn` for the space ARN. Because the context is an ARN, you can grant on `arn:aws:cloudwatch:<region>:<account-id>:space/*` before you create a space, then narrow to a specific space afterward.

2. **`kms:DescribeKey` for the principal that configures the key.** Grant this in a statement of its own, with no encryption context condition. `DescribeKey` requests do not carry an encryption context, so this action cannot sit under the condition that scopes the crypto grants.
3. **Crypto operations on evaluation metadata.** When you configure a customer managed key for evaluation metadata, the service requires `kms:Decrypt`, `kms:DescribeKey`, and `kms:GenerateDataKey` on that key.

Scope of the dataset key: AssociateDatasetKmsKey and DisassociateDatasetKmsKey operate on the Dataset, which holds your space's telemetry. The key is associated at the Dataset level rather than per signal type, so associating a customer managed key covers everything the Dataset holds, and disassociating it reverts all of it.

Sample key policy

The following sample key policy covers every way CloudWatch Omni uses your key: the space and Dataset checks described above, the material the service encrypts on your behalf, forwarded telemetry entering the Dataset, and reads by the space operator role. Add the statements you need to your key policy. Replace `111122223333` with your account ID and `us-east-1` with your Region.

```
{
  "Version": "2012-10-17",
  "Id": "omni-cmk-key-policy",
  "Statement": [
    {
      "Sid": "EnableRootKeyAdministration",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "AllowCloudWatchIntelligenceDescribeKey",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
```

```
      },
      "Action": "kms:DescribeKey",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    },
    {
      "Sid": "AllowCloudWatchIntelligenceEncryptionService",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Encrypt",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:cloudwatch:account": "111122223333"
        }
      }
    },
    {
      "Sid": "AllowOmniIntelligenceBranchKey",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws-crypto-ec:aws:cloudwatch:account": "111122223333"
```

```
        }
      }
    },
    {
      "Sid": "AllowCloudWatchDatasetDescribeKey",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": "kms:DescribeKey",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:dataset/default"
        }
      }
    },
    {
      "Sid": "AllowCloudWatchDatasetEncryption",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333",
          "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:dataset/default"
        }
      }
    },
```

```
    {
      "Sid": "AllowCloudWatchSpaceDescribeKey",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": "kms:DescribeKey",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:space/*"
        }
      }
    },
    {
      "Sid": "AllowCloudWatchSpaceEncryptionService",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudwatch.amazonaws.com"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "StringLike": {
          "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:space/*"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:space/*"
        }
      }
    },
    {
      "Sid": "AllowCallerDatasetForwarding",
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/service-role/
CloudWatchOmniDatasetIntegrationExecutionRole-1234567890123"
      },
      "Action": [
        "kms:GenerateDataKey",
        "kms:Encrypt",
        "kms:Decrypt"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default",
          "kms:ViaService": "cloudwatch.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowOperatorRoleDecrypt",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/service-role/
CloudWatchOmniSpaceAccessRole-1234567890123"
      },
      "Action": "kms:Decrypt",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "cloudwatch.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowOperatorRoleIntelligenceDecrypt",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/service-role/
CloudWatchOmniSpaceAccessRole-1234567890123"
      },
      "Action": "kms:Decrypt",
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "kms:EncryptionContext:aws-crypto-ec:aws:cloudwatch:account": "111122223333"
        }
      }
    }
  ]
}
```

The role ARNs in this sample use the form that setup creates in a single account: the `service-role/` path and a numeric suffix on the role name. Replace each one with the exact ARN of the role in your account, which you can copy from the IAM console. For an organization domain, the space operator role is `CloudWatchOmniOperatorRole` at the root path, with no suffix.

What each group of statements is for:

- **Root key administration** keeps your account able to manage the key. Do not remove it.
- **The `cloudwatch.amazonaws.com` statements** let the service encrypt and read your space content, your Dataset, and the analysis material the service builds from your telemetry. The encryption-context and `aws:SourceArn` conditions restrict each statement so the key can be used only for CloudWatch Omni resources in your own account.
- **`AllowCallerDatasetForwarding`** lets the Dataset integration execution role write forwarded logs and traces into a Dataset encrypted with this key. If you created your own execution role, use its ARN here. See the section called “Send telemetry to CloudWatch Omni”.
- **The operator-role statements** let the space operator role read data encrypted under the key, only through CloudWatch (`kms:ViaService`).

Encryption in transit

All communication with CloudWatch Omni service endpoints uses HTTPS with TLS. The public API endpoints require a minimum of TLS 1.2. We recommend TLS 1.3, and the service supports post-quantum hybrid key exchange. Amazon S3 access is TLS-enforced.

Telemetry ingestion is over HTTPS and is authenticated with Signature Version 4. Unsigned requests are rejected with an HTTP 403 response.

Important

CloudWatch Omni does not offer FIPS 140-3 validated endpoints. If you require FIPS 140-3 validated cryptographic modules, review this limitation before you send telemetry.

Operational data that CloudWatch Omni records about you

Request logs and audit events are delivered to AWS CloudTrail as management events, and include the resource ARN, the calling principal, the account ID, and the organization ID. Service metadata such as timestamps and resource identifiers is encrypted with AWS managed keys.

Identity and access management for CloudWatch Omni

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CloudWatch Omni resources. IAM is an AWS service that you can use with no additional charge.

CloudWatch Omni uses the Amazon CloudWatch identity. Actions are in the `cloudwatch` namespace and requests are signed with the `cloudwatch` Signature Version 4 signing name. The endpoint prefix differs from the signing name: requests go to `cloudwatch-omni.{region}.api.aws` but sign as `cloudwatch`.

Topics

- How CloudWatch Omni works with IAM
- Domains and spaces
- AWS managed policies
- Service roles used for setup
- Tag-based access control
- Telemetry ingestion identity
- Read-time access scope
- Space-level access control
- Troubleshooting identity and access

How CloudWatch Omni works with IAM

IAM feature	CloudWatch Omni support
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Forward access sessions	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how CloudWatch Omni and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Resource-based policies. CloudWatch Omni does not support resource-based policies. Space membership and access grants control fine-grained access within a space instead. See the section called "Control access to your space" and the section called "Access profiles".

Policy actions. Administrators can use AWS JSON policies to specify who has access to what: which principal can perform actions on which resources, and under what conditions. To see the list of CloudWatch Omni actions, see the *Service Authorization Reference*.

Policy resources. CloudWatch Omni supports the following resource types in policies:

`access-grant`, `access-profile`, `alert`, `dataset`, `domain`, `integration`, `omni-dashboard`, `space`, and `view`.

Domains and access grants that are scoped to an organization use the `organization-domain` and `organization-access-grant` resource types.

Policy condition keys. CloudWatch Omni supports the tag-based condition keys `aws:RequestTag/*` and `aws:TagKeys` on the operations that create a taggable resource, and `aws:ResourceTag/*` on tagged resources. See the "Tag-based access control" section of this page.

ACLs. Access control lists control which principals have permissions to access a resource. CloudWatch Omni does not support ACLs.

Attribute-based access control. Attribute-based access control (ABAC) defines permissions based on attributes, which in AWS are tags. CloudWatch Omni supports applying tags when you create a resource, and supports the tag-based condition keys.

Temporary credentials. CloudWatch Omni works with temporary credentials, and requests are signed with Signature Version 4. You can sign in with IAM Identity Center or with IAM. In both cases, CloudWatch Omni issues credentials that are scoped to the space you are working in.

Forward access sessions. When you run a telemetry query, CloudWatch Omni uses forward access session credentials to read your telemetry on your behalf, so the read is authorized as you rather than as the service. See the section called “Data encryption in CloudWatch Omni” for how those credentials are protected.

Service roles. CloudWatch Omni uses IAM roles that it assumes to read your telemetry and to operate on your resources. It creates most of them for you, and accepts a role that you supply for an organization domain and for agent evaluations. See the "Service roles used for setup" section of this page.

Service-linked roles. CloudWatch Omni uses a service-linked role, `AWSServiceRoleForCloudWatch_Omni`, which carries the `AWSCloudWatchOmniServiceRolePolicy` managed policy. It assumes that role to create the setup roles it manages in your account, and to attach their policies.

Domains and spaces

A *domain* and a *space* are distinct resources. A domain is the identity and federation boundary, and single sign-on is configured for each domain. A space is the top-level container for observability data within a domain, with the resource type `space` and the ARN form `space/{spaceId}`.

AWS managed policies

The space operator role carries `CloudWatchOmniSpaceAccessPolicy` as its base policy, and can also carry two optional policies chosen during setup: `CloudWatchOmniModelInferencePolicy` (described below) and `CloudWatchOmniAWSIntegrationPolicy`, which lets CloudWatch Omni discover resources and their relationships for Context Graph. CloudWatch Omni provides further managed policies for other roles, including `CloudWatchOmniDomainAccessPolicy` for organization domain administration. For the complete set and each policy's contents, see the section called “AWS managed policies for CloudWatch Omni”.

Managed policy	What it grants	When it is attached
`CloudWatchOmniSpaceAccessPolicy`	The space experience — space and access-grant management, telemetry queries, the context graph, alerts, dashboards, access profiles, views, threads, and integrations — together with evaluation management. It also carries the supporting permissions those need in other services, including CloudWatch Logs, AWS KMS, Secrets Manager, IAM Identity Center, and Amazon SNS.	Always. This is the base policy.
`CloudWatchOmniModelInferencePolicy`	Model inference only: invoking foundation models and inference profiles in Amazon Bedrock, including streaming responses, and invoking models served through Amazon Bedrock Mantle. It grants nothing else.	Only for spaces that use the prompt playground or run evaluations with custom evaluators. A view-only space does not receive it.

Keeping inference in its own policy means that a space can set up and view its evaluation configuration without holding permission to invoke a model. Only running a model requires the second policy.

When the CloudWatch Omni service-linked role creates the space operator role, it applies `CloudWatchOmniSpaceAccessPolicy` as the role's permissions boundary, which caps the role's effective permissions to that same set.

For what each policy grants action by action, the permissions that read metadata across your whole account, and the permissions you must provide yourself, see the section called “IAM policies to use CloudWatch Omni”.

Service roles used for setup

Setup uses several IAM roles, and CloudWatch Omni creates most of them for you. When you enable a capability, the service assumes its service-linked role and creates the role in your account with the matching AWS managed policy attached. Two roles are yours to create and pass in: the domain access role for an organization domain, and the evaluation execution role for online agent evaluations. You can also supply your own space operator role instead of letting CloudWatch Omni create one.

Role	Purpose	Created by	Required
`CloudWatchOmniSpaceAccessRole`, or `CloudWatchOmniOperatorRole` for an organization domain	The space operator role, which is the primary setup and data-access role. It carries the CloudWatch Omni actions along with the Amazon Bedrock, AWS KMS, AWS Config, Secrets Manager, CloudWatch Logs, and `iam:PassRole` permissions that setup and evaluation require. In a single	CloudWatch Omni, or you	Required

Role	Purpose	Created by	Required
	account, setup adds a numeric suffix to the role name. For an organization domain, one role per account is shared by every space in it. In the API it is the `dataAccessRoleArn` parameter.		
`CloudWatchOmniDomainAccessRole`	Domain access. You pass its ARN when you create a domain for your organization.	You	Required for an organization domain
`CloudWatchOmniIntegrationEnablementRole`	Enabling telemetry integrations for your account. One role per member account.	CloudWatch Omni	Required for integrations
`CloudWatchOmniDatasetIntegrationExecutionRole`	Dataset integration. Assumed by `logs.amazonaws.com`, and restricted to dataset integrations in your own account.	CloudWatch Omni	Required for dataset integration
`AgentCoreEvaluationRole`	Execution role for online agent evaluations. You pass its ARN when you create or update a space.	You	Optional. You can opt out, or point to an existing role.

Required means the role must exist for that capability to work, not that you have to create it yourself.

The operator role trusts the CloudWatch Omni service principal `cloudwatch.amazonaws.com`. For the trust policies that setup uses, see the section called "Cross-service confused deputy prevention".

Several operations take a role ARN that you supply, including your own operator role, the domain access role, and the evaluation execution role. Your calling principal needs `iam:PassRole` for the role you pass.

Tag-based access control

CloudWatch Omni supports tags on 11 resource types: `space`, `domain`, `organization-domain`, `access-grant`, `organization-access-grant`, `access-profile`, `omni-dashboard`, `alert`, `view`, `integration`, and `dataset`. You cannot tag any other CloudWatch resource through CloudWatch Omni.

Eight of these accept tags at creation: `space`, `domain`, `organization-domain`, `access-grant`, `organization-access-grant`, `access-profile`, `omni-dashboard`, and `alert`. For these, you can use `aws:RequestTag/*` and `aws:TagKeys` in your policies to control what a principal is allowed to create.

The remaining three (`view`, `integration`, and `dataset`) are tagged after they exist, so the request-tag condition keys do not apply to them.

You can use `aws:ResourceTag/*` to control access to any tagged resource.

Telemetry ingestion identity

You can authenticate telemetry ingestion in two ways. Both are available for metrics, logs, and traces.

1. **Signature Version 4 with an IAM role.** Sign OTLP ingestion requests with Signature Version 4 using temporary role credentials. This is the recommended approach.
2. **A bearer token.** For clients that cannot sign with Signature Version 4, you can use an IAM service-specific credential on a dedicated IAM user. This is available for metrics, logs, and traces, with a separate API key for each.

For metrics and logs, you create an API key from CloudWatch settings in the console. The console creates a dedicated IAM user with the `CloudWatchAPIKeyAccess` managed policy and issues the bearer token as a service-specific credential on that user. For the published setup steps, see Bearer token authentication for OTLP metrics in the *Amazon CloudWatch User Guide* and OTLP endpoints for CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*. For traces, see Setting up bearer token authentication for traces in the *X-Ray Developer Guide*.

You create and manage API keys and their credentials in the CloudWatch console using your own IAM permissions. CloudWatch Omni itself can list them but cannot create, rotate, or delete them.

For credential hygiene guidance, see the section called “Security best practices for CloudWatch Omni”.

Read-time access scope

A member's grant controls what their queries and views return. CloudWatch Omni applies the scope when it runs the query. To set it, see the section called “Limit what members can see”.

The scope constrains the telemetry query path. Other features can reach the same content, so grant their permissions with that in mind. See the section called “Security best practices for CloudWatch Omni”.

Space-level access control

IAM controls who can reach CloudWatch Omni and what the service may do in your account. Access to the contents of a space is a second, separate decision, evaluated inside the space. Both apply to every request.

- **The member's permission level** decides whether a principal may ask for an operation. A grant carries one of four levels: Viewer, Editor, Space Admin, or Custom, where Custom allows exactly the actions it lists.
- **The space operator role** decides whether the space may carry that operation out against your AWS resources. Raising a member to Space Admin adds no permission that this role does not already hold, so the role is the ceiling for everything in the space. One operator role serves every space in the account, so that ceiling is account-wide. You cannot separate two spaces from each other by giving them different operator roles.

For an audit, the properties that follow from that design:

- **Grants only add.** A principal's effective access is the union of every grant it holds, directly and through groups. For the tiered levels the highest one wins, and a Custom grant adds the actions it names. There are no explicit denies, so you cannot subtract access from a principal by adding a grant. You remove the grant that provides the access. Action scopes narrow only the actions they name, so they do not turn a tiered level into an exact allowlist. See the section called "Custom grant actions".
- **Read scope is row-level and enforced at read time.** A grant can carry a data scope that filters telemetry rows on the query path, before results reach the member. It filters rows, not fields: a member who can see a record sees every field on it.
- **A data scope on one grant does not narrow another.** A second grant without a scope restores the full view, including a grant inherited through a group.
- **Space administration is reserved for people.** An access profile, which is the identity a resource such as an alert uses when no one is signed in, can hold Viewer, Editor, or Custom, never Space Admin, and can never exceed the space operator role.
- **A space always retains at least one Space Admin.** The grant that provides the last Space Admin access cannot be deleted, and no principal can edit or delete its own grants.
- **Some grants are created by the service.** Those are marked service managed and cannot be edited or deleted.

To review or change any of this, see the section called "Control access to your space" for the model, the section called "Manage space members and permissions" for the members and grants surface, the section called "Limit what members can see" for data scopes, and the section called "Access profiles" for resource identities.

Troubleshooting identity and access

A `403` response with `AccessDeniedException` indicates that the calling principal is not authorized, or that the request signature did not validate. Confirm which identity the request is using with `aws sts get-caller-identity`, and confirm that the identity holds an active access grant in the space. A sign-in problem, such as an expired one-time code, returns a `400` rather than a `403`.

For signing telemetry ingestion requests from an OpenTelemetry exporter, see the section called "Send telemetry to CloudWatch Omni".

Compliance validation for CloudWatch Omni

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs. To learn whether CloudWatch Omni is within the scope of specific compliance programs, see AWS Services in Scope by Compliance Program and choose the compliance program that you are interested in.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using CloudWatch Omni is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and compliance quick start guides – Deployment guides that discuss architectural considerations and provide steps for deploying baseline environments on AWS.
- AWS compliance resources – A collection of workbooks and guides that might apply to your industry and location.
- AWS Config – A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – A service that provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

For general information, see AWS Compliance Programs.

> **Note**
>
> CloudWatch Omni is not in scope for any AWS compliance program. For the current status of every AWS service, see AWS Services in Scope by Compliance Program.

Resilience in CloudWatch Omni

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones

without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in CloudWatch Omni

As a managed service, CloudWatch Omni is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CloudWatch Omni through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests. CloudWatch Omni uses Signature Version 4 with the `cloudwatch` signing name.

When CloudWatch Omni reads your telemetry to answer a query, it uses forward access session credentials, so the read is authorized as you rather than as the service. For that identity model, see the section called “Identity and access management for CloudWatch Omni”, and for how those credentials are protected, see the section called “Data encryption in CloudWatch Omni”.

Tenancy and data segregation

CloudWatch Omni stores its metadata in shared, multi-tenant tables. Each item is logically isolated and scoped to the account that owns it. The underlying stores are not physically isolated for each account.

A request for a resource that another account owns is denied, and CloudWatch Omni does not reveal whether that resource exists, so resources cannot be enumerated across accounts. Resource-name uniqueness is scoped to each account.

The local trust boundary

Parts of CloudWatch Omni run on your own machine: the IDE extension, a local development server, and a local Model Context Protocol (MCP) server. Local mode is available in the IDE extension only and does not require an AWS connection. For what the extension does and how to set it up, see the section called “Develop with the IDE extension”.

You are responsible for the security of these local components and of the machine they run on:

- **The local MCP server has no authentication or authorization.** It runs as a child process of the client that launches it, and its proxy accepts connections only from the local machine. Any process that can run on your machine can access it.
- **The tooling can install its MCP configuration into third-party AI coding-assistant directories.** A connected assistant can then read raw local trace content, including prompts, responses, and tool inputs and outputs, without any additional authorization. Before you install the configuration, decide whether that assistant should have access to this data.
- **Local telemetry files are plaintext.** The service does not encrypt them. See the section called “Data encryption in CloudWatch Omni”.

Code-based evaluators run as AWS Lambda functions in your own account. CloudWatch Omni invokes a function whose name begins with `cloudwatchCodeEvaluator` and reads its configuration; it cannot create, modify, or delete the function. Evaluator code runs under an execution role that you control, and CloudWatch Omni does not host or execute evaluator code in its own service. Prompt-based evaluators run in Amazon Bedrock AgentCore and are scored by the model you configure. See the section called “Evaluators and evaluations”.

Interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your virtual private cloud (VPC) and CloudWatch Omni by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, which you can use to privately access CloudWatch Omni without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC do not need public IP addresses to communicate with the service, and traffic between your VPC and CloudWatch Omni does not leave the AWS network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets.

For more information, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

Endpoint services for CloudWatch Omni

CloudWatch Omni offers two VPC endpoint services. Create an interface endpoint for each service that you use.

Endpoint service name	Private DNS name	What it carries
`com.amazonaws.{region}.cloudwatch-omni`	`cloudwatch-omni.{region}.api.aws`	The CloudWatch Omni API and telemetry ingestion
`com.amazonaws.{region}.cloudwatch-omni-threads`	`cloudwatch-omni-threads.{region}.api.aws`	The WebSocket session that the Omni agent uses, on port 8443

Both endpoint services support private DNS and VPC endpoint policies. Because each private DNS name is the same as the corresponding public hostname, a client in a VPC that has the interface endpoint reaches the service privately with no change to its configuration.

Considerations for CloudWatch Omni VPC endpoints

Before you set up an interface VPC endpoint for CloudWatch Omni, review Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

- The endpoint provides private access to the CloudWatch Omni service endpoint `cloudwatch-omni.{region}.api.aws`, for both the API and telemetry ingestion. All communication uses TLS. The endpoint requires a minimum of TLS 1.2 and supports TLS 1.3 with post-quantum hybrid key exchange.
- The endpoint is a dual-stack endpoint, supporting both IPv4 and IPv6.
- CloudWatch Omni supports private DNS for the endpoint. When you enable private DNS, requests to the default service hostname `cloudwatch-omni.{region}.api.aws` resolve to your interface endpoint automatically, so you do not have to change your client endpoint configuration.

- CloudWatch Omni supports VPC endpoint policies. See the "Create a VPC endpoint policy for CloudWatch Omni" section of this page.
- The endpoint is available in the AWS Regions where CloudWatch Omni is available.

Create an interface VPC endpoint for CloudWatch Omni

You can create an interface VPC endpoint for CloudWatch Omni using either the Amazon VPC console or the AWS Command Line Interface. For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface VPC endpoint for CloudWatch Omni using the following service name:

```
com.amazonaws.{region}.cloudwatch-omni
```

For example, in US East (N. Virginia):

```
com.amazonaws.us-east-1.cloudwatch-omni
```

If you enable private DNS for the endpoint, you can make API and telemetry-ingestion requests to CloudWatch Omni using its default DNS name for the Region, `cloudwatch-omni.{region}.api.aws`, and they route through the interface endpoint.

Create a VPC endpoint policy for CloudWatch Omni

You can attach an endpoint policy to your VPC endpoint that controls access to CloudWatch Omni. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

By default, full access to CloudWatch Omni is allowed through the endpoint. To restrict access, attach a custom endpoint policy to the interface endpoint. For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Configuration and vulnerability analysis in CloudWatch Omni

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

For the managed service, AWS handles basic security tasks such as guest operating system and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following AWS resources:

- Shared responsibility model
- Best practices for security, identity, and compliance

CloudWatch Omni also ships software that you install. The following components run in your environment, and keeping them current is your responsibility. For the extension itself, see the section called "Develop with the IDE extension". For installing the collector, see the section called "Send telemetry to CloudWatch Omni".

Component	How it updates	Your responsibility
The IDE extension	Updates itself through the extension gallery.	Keep automatic updates enabled, and apply updates.
The local MCP server	Bundled inside the IDE extension and versioned with it. It has no separate patch cadence.	Update it by updating the extension.
The local development server	Bundled with the IDE extension.	Update it by updating the extension.
The AWS Distro for OpenTelemetry collector	You install it and you patch it. It is pinned in your own dependency manifests.	Track and apply upstream security updates yourself.

Updates and vulnerability notification. There is no separate security-bulletin or vulnerability-notification channel for these components. Keep extension updates enabled, and track upstream OpenTelemetry releases yourself to stay current on security fixes.

The local MCP server and third-party AI assistants

The local MCP server has no authentication. See the section called “Infrastructure security in CloudWatch Omni”. Its tool set exposes trace search, dataset, evaluation, and credential operations. Because the tooling can connect this content to third-party AI coding assistants, review and control which assistants you allow to access your local trace and dataset content. See the section called “Security best practices for CloudWatch Omni”.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that does not have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that CloudWatch Omni gives another service to the resource. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

CloudWatch Omni assumes IAM roles in your account to read your telemetry, both during setup and during evaluation. To prevent it from being used as a confused deputy, restrict the roles you create so that they can only be assumed on your behalf.

Setup and data-access roles

The space operator role trusts the CloudWatch Omni service principal `cloudwatch.amazonaws.com`. When you set up a space in a single account, CloudWatch Omni creates this role and applies the following trust policy. If you supply your own role instead, give it the same trust policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "cloudwatch.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession", "sts:SetSourceIdentity",
 "sts:SetContext"],
      "Condition": {
        "StringEquals": { "aws:SourceAccount": "<YOUR_OWN_ACCOUNT_ID>" }
      }
    }
  ]
}
```

The `aws:SourceAccount` condition limits CloudWatch Omni to assuming this role on behalf of your own account.

Dataset integration execution role

Dataset integration reads your CloudWatch Logs data through a role that CloudWatch Logs assumes on your behalf. Because the calling service is CloudWatch Logs rather than CloudWatch Omni, this role's trust policy uses both `aws:SourceAccount` and `aws:SourceArn` to restrict the assumption to dataset integrations in your own account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "logs.amazonaws.com" },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": { "aws:SourceAccount": "<YOUR_OWN_ACCOUNT_ID>" },
        "ArnLike": { "aws:SourceArn":
 "arn:aws:observabilityadmin:*:<YOUR_OWN_ACCOUNT_ID>:dataset-integration/*" }
      }
    }
  ]
}
```

Security best practices for CloudWatch Omni

CloudWatch Omni provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and do not represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Preventative controls

Use temporary credentials for telemetry ingestion. Prefer Signature Version 4 with temporary role credentials over long-lived access keys. Never embed credentials in source code or in client applications. Store any secrets in a secrets manager and rotate them regularly.

Prefer role credentials over bearer tokens. For metrics, logs, and traces, telemetry ingestion supports both Signature Version 4 with an IAM role and a long-lived bearer token, which is an IAM user service-specific credential, with a separate one for each signal. Prefer the role path. If you use a bearer token, scope it so that it can only send telemetry, store it in a secrets manager, and rotate it regularly. See the section called “Identity and access management for CloudWatch Omni”.

Apply least privilege and keep the administrator group small. Grant each member the lowest permission level that lets them do their job. Keep the set of space administrators small, and review access periodically.

Restrict roles that call CloudWatch Omni outside the console. When a member works in a space, CloudWatch Omni issues credentials for that space and the member's access is limited to what their grants allow. When a principal calls with its own IAM credentials (an automation role, a build pipeline, or an SDK client), a missing grant does not deny the request. The request is authorized by that principal's IAM policy instead. Grant each such role only the actions it needs, and do not rely on space grants to constrain it. See the section called “Identity and access management for CloudWatch Omni”.

Require a grant on roles you create for programmatic access. To make a role's CloudWatch Omni permissions effective only when the caller has a grant in the space, add the condition `"cloudwatch:HasAccessGrant": "true"` to the role's policy. Keep the setup and administration actions in a separate statement without the condition, because those operations are authorized by IAM alone.

Use a Custom grant when you need an exact set of actions. Adding action scopes to a grant on one of the tiered permission levels narrows only the actions you name; every other action that the

level allows remains available. If your intent is to allow an exact set of actions and nothing else, use a Custom grant, which is an allowlist. See the section called “Custom grant actions”.

Redact sensitive content at capture time. CloudWatch Omni does not automatically detect or redact personally identifiable information. Redacting before content leaves your application process is the only layer that guarantees data never leaves your application. For the mechanisms available at each point in the pipeline, see the section called “Protect sensitive data”.

Choose your judge model with your compliance constraints in mind. Restrict which telemetry you evaluate and choose the judge model according to where trace content is allowed to flow. For what a judge model receives, see the section called “Protect sensitive data”. For cross-Region inference, see the section called “Cross-Region inference in CloudWatch Omni”.

Constrain access at the dataset and export layer, not only with query filters. Data scopes do not extend to dataset creation and export, to AI-assisted summaries, or to prompt playground runs. A member who is restricted from a field in queries could still reach that content by building a dataset or running an AI-assisted summary over the underlying traces. Grant dataset, evaluation, export, and AI-feature permissions with this in mind.

Do not rely on data scopes for metrics. A data scope filters log and trace rows. It does not filter metrics. If a member should not see a metric, do not grant them the metric-read actions.

Review member data scopes when you deploy new agents. A data scope is an inclusion rule: it lists the values a member is allowed to see, so a newly deployed agent is not visible to a scoped member until you add it to their scope. Review data scopes whenever you deploy a new agent, so that members who need the new agent can see it.

Verify data scope changes. Saving an edit replaces the member's previous scope rather than adding to it. After you change a member's data scope, have the member run a query to confirm the result.

Review AI-proposed instrumentation changes. Treat AI-generated instrumentation diffs like any other code change, and review them before you apply them.

Control local content exposure. Local `.omni/` files are plaintext, and the local MCP server is unauthenticated. The tooling can share local trace content with third-party AI assistants, so restrict which assistants you connect, and keep raw trace files out of source control. See the section called “Configuration and vulnerability analysis in CloudWatch Omni”.

Detective controls

Audit management activity with AWS CloudTrail. Management operations are recorded in CloudTrail with the event source `cloudwatch.amazonaws.com`. Monitor critical operations and alarm on anomalies.

Opting out of using your data for service improvement

CloudWatch Omni does not use your content to develop or improve the service or its AI features, and your content is not used to train any foundation models. Your content is used only to provide the service to you.

Some CloudWatch Omni features send your content to other AWS services to process. Those services have their own policies, and for some of them you can opt out of having your content used for service improvement. This page describes what each service does and how to opt out where an opt-out is available.

How CloudWatch Omni uses your content

CloudWatch Omni processes the telemetry you send, and your input to its AI features, only to provide the service to you: to answer your questions, run your queries and evaluations, and render your dashboards. It does not retain that content for service improvement, and it does not use it for model training.

For where those requests are processed, and the Regions they can reach, see the section called "Cross-Region inference in CloudWatch Omni".

Other AWS services that process your content

Several features send your content to another AWS service. CloudWatch Omni makes those calls using your own AWS credentials, in your own AWS account.

Feature	Content sent	AWS service
The Omni agent, AI-assisted dashboard creation, and AI-assisted summaries	Your questions and the responses, and the telemetry from your space that is used to answer them	Amazon Bedrock

Feature	Content sent	AWS service
Prompt playground	The prompt you enter, any context you attach, and the model response	Amazon Bedrock
Evaluations	The agent input, output, and trace content that is scored, and the evaluation instructi ons	Amazon Bedrock, Amazon Bedrock AgentCore
Voice input and voice playback	The audio you record, its transcription, and the text that is read back to you	Amazon Transcribe, Amazon Polly

Amazon Bedrock

Amazon Bedrock does not use your inputs or outputs to train models or to improve the service, and does not share them with model providers. For more information, see the Amazon Bedrock FAQs. Amazon Bedrock is not one of the services covered by the AI services opt-out policy, because it does not use your content for service improvement.

For certain models, Amazon Bedrock retains model inputs and outputs for up to 30 days to detect abuse. Abuse detection is separate from service improvement and is not governed by the AI services opt-out policy. For more information, see Amazon Bedrock abuse detection in the *Amazon Bedrock User Guide*.

Amazon Bedrock AgentCore

Amazon Bedrock AgentCore processes content for evaluations that run against an agent runtime. AgentCore states that it "may use and store your content to improve your service experience or performance," and that "such improvements would be for your use of AgentCore and not for other customers." For more information, see What is Amazon Bedrock AgentCore in the *Amazon Bedrock AgentCore Developer Guide*.

AgentCore is not covered by the AI services opt-out policy.

Amazon Transcribe and Amazon Polly

Amazon Transcribe and Amazon Polly process the audio and text for voice input and voice playback. Both services can use content for service improvement, and both are covered by the AI services opt-out policy. To opt out for these services, create an AI services opt-out policy that includes the `transcribe` and `polly` service keys.

```
{
    "services": {
        "transcribe": {
            "opt_out_policy": {
                "@@assign": "optOut"
            }
        },
        "polly": {
            "opt_out_policy": {
                "@@assign": "optOut"
            }
        }
    }
}
```

For instructions, see AI services opt-out policies in the *AWS Organizations User Guide*.

Opting out for other Amazon CloudWatch AI features

Other Amazon CloudWatch features that use artificial intelligence (including natural language query generation for CloudWatch Metrics Insights and CloudWatch Logs Insights, and CloudWatch investigations) can use your content for service improvement. Those features are covered by the AI services opt-out policy under the `cloudwatch` service key. Opting out with that key does not change CloudWatch Omni, which does not use your content for service improvement.

```
{
    "services": {
        "cloudwatch": {
            "opt_out_policy": {
                "@@assign": "optOut"
            }
        }
    }
}
```

> **Note**
>
> An AI services opt-out policy is set in AWS Organizations, not in CloudWatch. Your organization must have all features enabled, and you must enable the AI services opt-out policy type before you can attach a policy. The policy applies to every account in the organization or organizational unit that you attach it to. A standalone account that is not part of an organization cannot use an AI services opt-out policy.

Limiting what enters your telemetry

Whether a service may use your content for improvement is separate from what your telemetry contains. To keep model prompt and response text out of your traces entirely, configure your instrumentation to omit it. See the section called “Protect sensitive data”.

IAM policies to use CloudWatch Omni

The space operator role carries `CloudWatchOmniSpaceAccessPolicy` as its base policy, with `CloudWatchOmniModelInferencePolicy` and `CloudWatchOmniAWSIntegrationPolicy` as optional additions chosen during setup. Between them they cover using a space: querying telemetry, managing alerts and dashboards, and creating and running evaluations. None covers creating a space or a domain, so setup needs permissions you grant yourself. CloudWatch Omni provides further managed policies for other roles, including `CloudWatchOmniDomainAccessPolicy` for organization domain administration. For the complete set, see the section called “AWS managed policies for CloudWatch Omni”.

This page enumerates what each policy grants and why, discloses the permissions that read metadata across your whole account, and lists the permissions you must add separately. For how CloudWatch Omni works with IAM in general (resource types, condition keys, and the roles setup creates), see the section called “Identity and access management for CloudWatch Omni”.

Topics

- Permissions in CloudWatchOmniModelInferencePolicy
- Permissions in CloudWatchOmniSpaceAccessPolicy
- Permissions that read across your whole account
- Permissions you provide yourself

Permissions in CloudWatchOmniModelInferencePolicy

This policy grants model inference and nothing else. It is attached only to spaces that use the prompt playground or run evaluations that invoke a model. A space that only views telemetry does not receive it, and a space can create and view its evaluation configuration without it. Only running a model requires it.

Action	Resources	Why CloudWatch Omni needs it
`bedrock:InvokeModel` , `bedrock:InvokeModelWithResponseStream`	Foundation models and inference profiles	Runs the model you select in the prompt playground, and the model an evaluator uses to score a result.
`bedrock-mantle:CreateInference`	Amazon Bedrock Mantle projects	Runs a model served through Amazon Bedrock Mantle rather than as a Bedrock foundation model.
`bedrock-mantle:CallWithBearerToken`	All resources	Authenticates the Amazon Bedrock Mantle inference call above.

Every action except `bedrock-mantle:CallWithBearerToken` is restricted to your own account by an `aws:ResourceAccount` condition, so a space can invoke only the models, inference profiles, and projects in the account it belongs to. `CallWithBearerToken` has no resource type in the IAM action model, so it cannot be scoped to a resource or narrowed by a condition. The same action appears with the same unscoped resource in the AWS managed policy BedrockAgentCoreFullAccess, which uses it for the same purpose.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AgentCoreEvaluationBedrockInvoke",
            "Effect": "Allow",
            "Action": [
```

```
                "bedrock:InvokeModel",
                "bedrock:InvokeModelWithResponseStream"
            ],
            "Resource": [
                "arn:aws:bedrock:*::foundation-model/*",
                "arn:aws:bedrock:*:*:inference-profile/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "AgentCoreEvaluationBedrockMantleInference",
            "Effect": "Allow",
            "Action": "bedrock-mantle:CreateInference",
            "Resource": "arn:aws:bedrock-mantle:*:*:project/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Sid": "AgentCoreEvaluationBedrockMantleCallWithBearerToken",
            "Effect": "Allow",
            "Action": "bedrock-mantle:CallWithBearerToken",
            "Resource": "*"
        }
    ]
}
```

Permissions in CloudWatchOmniSpaceAccessPolicy

This is the base policy, always attached. It grants the space experience and the management side of evaluation. The table groups its permissions by what they are for, and gives the condition that limits each group.

Permission group	What it covers	How it is limited
The space experience, in the `cloudwatch` namespace	Space and access-grant management, telemetry query sessions, the context graph, alerts, dashboards, access profiles, views, user preferences, threads, integrations, intelligence configuration, and tagging.	Every action requires the condition `cloudwatch:HasAccessGrant` to be `true`, so a principal with no grant in the space cannot use any of them.
Looking up IAM Identity Center users and groups	`identitystore:ListUsers`, `identitystore:ListGroups`, `identitystore:DescribeUser`, and `identitystore:DescribeGroup` against your Identity Store directories, users, and groups, used when you search for a person or group to grant access to a space.	The call must arrive through CloudWatch (`aws:CalledVia` is `cloudwatch.amazonaws.com`), so the role cannot call Identity Store directly.
A deny on tagging outside CloudWatch Omni	An explicit `Deny` on the three tagging actions for every resource other than the CloudWatch Omni resource types.	Prevents a space session from tagging a classic CloudWatch resource, such as an alarm or a CloudWatch dashboard, to gain access at the resource level.
Passing a role to CloudWatch Omni	`iam:PassRole`, so you can hand CloudWatch Omni the service role you created for the space.	Same account, and `iam:PassedToService` must be `cloudwatch.amazonaws.com`. The role name cannot be restricted by prefix because you supply your own role.

Permission group	What it covers	How it is limited
Passing a role for online evaluations	`iam:PassRole` for the evaluation execution role, so an online evaluation can run when nobody is signed in.	Restricted to roles named `service-role/AgentCoreEvaluationRole*` in your own account, passed only to `bedrock-agentcore.amazonaws.com`.
Reading ingestion users	`iam:ListUserTags` and `iam:ListServiceSpecificCredentials`, so the console can show the bearer tokens that already exist.	Restricted to users under the `omni/` path named `OmniIngestionUser-*` in your own account. Read-only: CloudWatch Omni cannot create, rotate, or delete them.
Reading a customer managed key	`kms:DescribeKey`, so the console can validate and display a key you select.	Same account. Metadata only, and it does not permit any cryptographic operation.
Encrypting space content	`kms:Decrypt` and `kms:GenerateDataKey` against the key you chose for the space.	Four conditions together: same account, the key carries the `cw-omni` tag, the call arrives through CloudWatch, and the encryption context names a space ARN — so the key cannot be used for any other CloudWatch resource type.

Permission group	What it covers	How it is limited
Reading an encrypted IAM Identity Center directory	`kms:Decrypt` against the customer managed key that encrypts your IAM Identity Center directory, so the directory lookups above can read it.	The call must come through Identity Store (`kms:ViaService`), and the encryption context must include the `aws:identitystore:identitystore-arn` key, so it applies only to Identity Store data. It has no same-account condition, because the directory and its key are often in your organization's management account; the key policy remains the control.
Encrypting evaluator material	`kms:Decrypt` and `kms:GenerateDataKey` for evaluator and model-provider key material at the time it is used.	Same account, the request must arrive via `bedrock-agentcore.amazonaws.com`, and the encryption context must name an evaluator ARN in your own account.
Evaluators, datasets, and online evaluation configurations	Full lifecycle of the Amazon Bedrock AgentCore resources that back evaluation, including running an evaluation and tagging these resources.	Scoped separately to the `evaluator`, `dataset`, and `online-evaluation-config` resource types, each in your own account.

Permission group	What it covers	How it is limited
Listing the model catalog	`bedrock:ListFoundationModels` and `bedrock:ListInferenceProfiles`, so you can pick a model when you create an evaluator or open the playground.	Listing only. Invoking a model requires the separate inference policy above.
Writing evaluation results	Creating the log groups, log streams, and log events that record evaluation scores and annotations in CloudWatch Logs.	Log streams and events are written only under `/aws/cloudwatch/`; under `/aws/bedrock-agentcore/evaluations/` only the log group itself can be created. Both in your own account.
Choosing which telemetry an online evaluation reads	`logs:DescribeLogGroups` for the log group picker, and reading and appending a field index on the trace log group so sampling can filter spans by service name.	Same account. The picker lists any log group in the account, because telemetry does not always follow the `/aws/` naming conventions. Names only, not contents.
Running a code-based evaluator	`lambda:InvokeFunction` and `lambda:GetFunction` for evaluators you implement as code rather than as a prompt.	Restricted to functions named `cloudwatchCodeEvaluator*` in your own account.

Permission group	What it covers	How it is limited
Storing a model-provider API key	Creating, reading, updating the value of, deleting, and tagging the Secrets Manager secret that holds a model-provider API key you supply.	Restricted to secrets whose name begins `cw-omni-` that carry the `cw-omni` tag, in your own account. Creating one also requires the `cw-omni` tag on the request.
AWS Config resource inventory	The AWS Config service-linked configuration recorder, its status, and the AWS Config service-linked role, used to collect resource inventory for your account.	Same account, and the recorder's service principal must be `cloudwatch.amazonaws.com`. Creating the service-linked role is restricted to the `config.amazonaws.com` path.

Permissions that read across your whole account

Five permission groups grant permissions that the underlying API cannot restrict to a subset of resources, so they enumerate metadata across your entire account:

- `iam:ListRoles`, `iam:GetRole`, and `iam:ListAttachedRolePolicies` reach any IAM role in the account, including its details and the managed policies attached to it.
- `iam:ListUsers` reaches the name of any IAM user, not only the ingestion users.
- `secretsmanager:ListSecrets` reaches the name and metadata of every secret, not only the ones CloudWatch Omni creates.
- `sns:ListTopics` reaches the ARN of every Amazon SNS topic in the account, so you can choose a notification target for an alert. It returns topic ARNs only (no attributes, subscriptions, or messages).
- `bedrock:ListFoundationModels` and `bedrock:ListInferenceProfiles` list the models and inference profiles available to you.

None of them permits a change, and none reads the contents of anything. Apart from creating the AWS Config service-linked role, this policy grants no permission to create, modify, or delete an IAM role, user, or policy, and reading a secret value is a separate permission limited to secrets in your own account whose name begins with `cw-omni-` and that carry the `cw-omni` tag.

For the complete policy document, see CloudWatchOmniSpaceAccessPolicy in the *AWS Managed Policy Reference*.

Permissions you provide yourself

The managed policies cover using a space. They do not cover creating one, and they do not cover telemetry ingestion.

To do this	You need
Create a domain or a space	Setup actions that neither managed policy grants. See the section called “Set up Omni”.
Pass a role that you supply	`iam:PassRole` on your own calling principal for the role you pass, including your own operator role, the domain access role, and the evaluation execution role.
Send metrics or logs with a bearer token	Your own IAM permissions to create the ingestion user and its credential. The user carries `CloudWatchOmniOTLPPolicy`. Bearer tokens apply to metrics and logs only. Traces are signed with Signature Version 4. See the section called “Identity and access management for CloudWatch Omni”.
Use a customer managed key	A key policy that allows the space operator role to use the key, and the `cw-omni` tag on the key. See the section called “Data encryption in CloudWatch Omni”.

Cross-Region inference in CloudWatch Omni

CloudWatch Omni includes features that use artificial intelligence (AI), such as the Omni agent, AI-assisted dashboard creation, and evaluations that score agent behavior with a judge model. When you use one of these features, CloudWatch Omni sends data from your space, such as telemetry content and your natural language input, to a large language model. CloudWatch Omni uses cross-Region inference to distribute that traffic across AWS Regions.

Your data remains stored only in your space's Region. When you use a feature that uses AI, your input and output might be processed outside of your space's Region. All data is transmitted encrypted across Amazon's secure network, using a minimum of TLS 1.2.

For most space Regions, CloudWatch Omni uses geographic inference profiles, which keep processing within the geography that your space's Region belongs to:

- Requests from a space in the United States are processed within the United States.
- Requests from a space in Canada are processed within Canada or the United States.
- Requests from a space in Europe are processed within Europe. Europe (London) is a possible inference Region only for requests from a space in Europe (London).
- Requests from a space in Australia are processed within Australia.
- Requests from a space in Japan are processed within Japan.

For how to control what content reaches your telemetry, see the section called “Protect sensitive data”. For how evaluations use judge models, see the section called “Evaluators and evaluations”. For how CloudWatch Omni and the AWS services it calls on your behalf use your content, see the section called “Opting out of using your data for service improvement”.

Global cross-Region inference for specific Regions

For spaces in the following Regions, the geography-based routing described previously doesn't apply. Instead, CloudWatch Omni selects the optimal Region globally to process your requests.

- Asia Pacific (Mumbai) – `ap-south-1`
- Asia Pacific (Singapore) – `ap-southeast-1`
- South America (São Paulo) – `sa-east-1`

Where your requests are processed

The following table shows which Regions can process requests from each space Region. The possible inference Regions are the destinations of the cross-Region inference profile, and can include AWS Regions where CloudWatch Omni itself is not available.

Supported CloudWatch Omni geography	Space Region	Possible inference Regions
United States (US)	US East (N. Virginia)	US East (N. Virginia), US East (Ohio), US West (Oregon)
United States (US)	US West (Oregon)	US East (N. Virginia), US East (Ohio), US West (Oregon)
Canada	Canada (Central)	Canada (Central), US East (N. Virginia), US East (Ohio), US West (Oregon)
Europe (EU)	Europe (Frankfurt)	Europe (Frankfurt), Europe (Ireland), Europe (Milan), Europe (Paris), Europe (Spain), Europe (Stockholm)
Europe (EU)	Europe (Ireland)	Europe (Frankfurt), Europe (Ireland), Europe (Milan), Europe (Paris), Europe (Spain), Europe (Stockholm)
Europe (EU)	Europe (London)	Europe (Frankfurt), Europe (Ireland), Europe (London), Europe (Milan), Europe (Paris), Europe (Spain), Europe (Stockholm)
Australia	Asia Pacific (Sydney)	Asia Pacific (Melbourne), Asia Pacific (Sydney)

Supported CloudWatch Omni geography	Space Region	Possible inference Regions
Japan	Asia Pacific (Tokyo)	Asia Pacific (Osaka), Asia Pacific (Tokyo)
Global	Asia Pacific (Mumbai)	Selected globally
Global	Asia Pacific (Singapore)	Selected globally
Global	South America (São Paulo)	Selected globally

Models that you choose

You choose the model in the prompt playground and when you configure an evaluator. CloudWatch Omni lists the foundation models and cross-Region inference profiles that are available to you in your Region, and calls the model you select with your own credentials, in your own account.

The list includes global inference profiles. If you select a global inference profile, the optimal Region to process your request is selected globally rather than only within your geography.

To restrict which Regions can process these requests, use IAM permissions on inference profiles or service control policies in your own account. For the managed policy that grants model inference, see the section called “IAM policies to use CloudWatch Omni”.

Tutorial: Monitor an application

Use this tutorial to instrument an application with OpenTelemetry, send its telemetry to CloudWatch Omni, and verify that you can monitor its services and requests. To work through the AI agent workflow instead, see the section called “Tutorial: Fix a production quality regression”.

Prerequisites

- Set up a CloudWatch Omni domain and space. See the section called “Set up Omni”.
- An application serving traffic or requests. Use an OpenTelemetry SDK for your application language, or supported auto-instrumentation where applicable. For supported application languages, see the section called “Supported environments, languages, and frameworks”.

- Configure credentials that allow the application to send telemetry to CloudWatch. See the section called “Send telemetry to CloudWatch Omni”.

Step 1: Instrument your application

1. Add OpenTelemetry instrumentation: Instrument your application with the OpenTelemetry SDK for your language, or with supported auto-instrumentation. For how OpenTelemetry works with CloudWatch, see Getting started with OTLP ingestion.
2. Set the service and application identity: Set the OpenTelemetry service.name resource attribute for each service. You can set this value with the OTEL_SERVICE_NAME environment variable. Omni uses this value to identify the service. Set service.namespace on services that belong to the same application. Omni groups services with the same service.namespace value into the same application. Use consistent values across your telemetry so Omni can correlate signals from the same service.
3. Instrument outbound calls: Make sure your instrumentation covers the outbound operations that you want to see as dependencies, such as HTTP client requests, database calls, and AWS SDK calls. Omni uses spans from these calls to identify service dependencies. A call that is not instrumented might not appear as a dependency.
4. Configure service metrics: Add the CloudWatch plugin for OpenTelemetry to your instrumentation, following the optional OpenTelemetry instrumentation step in the section called “Send application telemetry”. The plugin generates request, error, and duration (RED) metrics from your application's spans, and Omni uses those metrics for service request, latency, and errors. For the package to install for your language and its documentation, see the section called “Monitor a service”. The plugin derives these metrics from your spans but does not export them itself. The metrics exporter in your OpenTelemetry configuration sends them, so configure a metrics exporter alongside your trace exporter.

Step 2: Send telemetry to CloudWatch

1. Configure the OpenTelemetry (OTLP) exporter to send traces to the CloudWatch endpoint for your Region, authenticated with AWS Signature Version 4 (SigV4). For the endpoint, the exporter configuration, and the permissions your application's role needs, follow the section called “Send telemetry to CloudWatch Omni”.
2. Run your application and send several requests through it. These requests generate telemetry that CloudWatch makes available to your Omni space. If telemetry does not appear, verify the

exporter endpoint, Region, credentials, and signing configuration. See the troubleshooting guidance in the section called “Send telemetry to CloudWatch Omni”.

Step 3: See your application on the map

1. Open Application map in Omni. Set the time range to include the requests that you generated. The application map reflects telemetry observed during the selected period. Find the application identified by the service.namespace value that you configured in step 1. The application contains the services that share that namespace. Each service is identified by its service.name.
2. Open the application to review its services and dependencies. Each service appears as a service node named with the service name you set in step 1. For more information, see the section called “Monitor applications with the application map”.

Step 4: Examine service health

1. Select a service from the application map. Review the available request, latency, and error information for the selected time range. You can also inspect the services and resources connected to it. For more information about finding and monitoring services, see the section called “Monitor a service”.
2. From the service, pivot into its traces to follow one request across your services, span by span. For the application traces, see the section called “Application traces”.

Next steps

- Query the telemetry behind the map with SQL. See the section called “Explore and query your telemetry”.
- Put your application's telemetry on a dashboard. See the section called “Dashboards”.
- Create alerts on your application's telemetry. See the section called “Alerts”.
- If your system includes AI agents, monitor them alongside your application. See the section called “Monitor AI agents”.

Tutorial: Fix a production quality regression

This tutorial takes an agent through the full lifecycle in CloudWatch Omni: instrument and test it in the IDE extension, deploy it and monitor it in the Omni web UI, catch a real quality problem on live traffic, reproduce and fix it against the exact failing runs, and set up a continuous check so the same problem cannot return unnoticed.

Start here if you are new to Omni. Every action is a step, including installing the extension, and each step ends with a checkpoint so you can tell whether it worked before moving on.

What the tutorial teaches is a method: define the failure precisely enough to score it, capture the exact runs that fail, fix them against an unchanged baseline, then keep scoring live traffic so the failure cannot come back quietly. The worked example is a **scope violation**: a customer support agent answers an off-topic question (a cookie recipe) that it should politely decline. It completes without errors, and general-purpose evaluators score it as perfectly helpful, which is why it survives ordinary monitoring.

Both surfaces work against the same space, which is what makes the handoff work: a dataset you package in the Omni web UI imports straight into the extension, and the evaluator that defines the failure scores identically during the fix and on live traffic afterward.

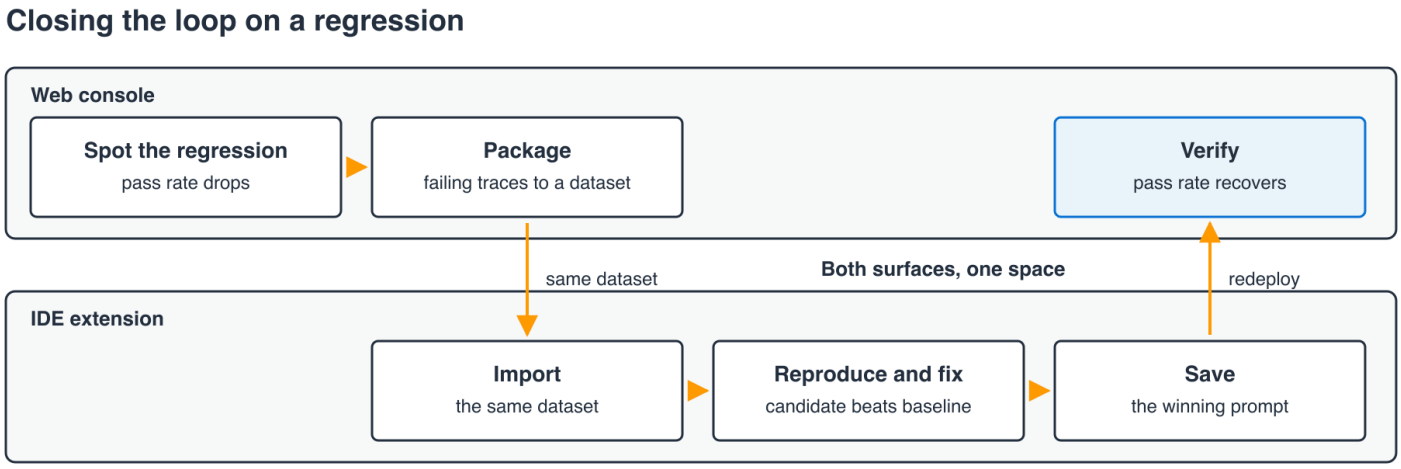
What you will do

Phase	Surface	Steps
A. Build and instrument	IDE extension	1–3. Install the extension, instrument your agent, and get your first traces locally.
B. Deploy and observe	Omni web UI	4–8. Deploy, drive traffic, monitor cost, find the failing run, and package it as a dataset.
C. Reproduce and fix	IDE extension	9–13. Import the dataset, write an evaluator that catches the failure, fix it

Phase	Surface	Steps
		against the baseline, and redeploy.
D. Verify continuously	Omni web UI	14. Score live traffic so the regression cannot return unnoticed.

Experiments and prompt management are available in the IDE extension only.

The following diagram shows the full loop across both surfaces. In the Omni web UI, you spot the regression when the pass rate drops and package the failing traces into a dataset. In the IDE extension, you reproduce and fix the problem until the candidate beats the baseline, then save the winning prompt. You then redeploy and return to the web UI to confirm the pass rate recovers. Both surfaces work against the same space.



Prerequisites

- A space in CloudWatch Omni, in the account and Region you will deploy into. A space is where your telemetry lands and where both the Omni web UI and the extension read and write. Creating one is a separate setup task. See the section called “Set up Omni”.

> **Important**
>
> Set up your space before you deploy in step 4. Traces your agent sends before span ingestion is enabled are never stored in CloudWatch, so they will not appear in your space later. See the section called “Set up Omni”.

- An agent. You can use the sample agent bundled with the extension, or your own. See **Choose your agent** below. Your own agent needs a supported framework, in Python or TypeScript. See the section called “Supported environments, languages, and frameworks”.
- Somewhere to deploy it. The sample deploys to Amazon Bedrock AgentCore, where the runtime turns on agent observability for you. Any runtime works. You configure the OpenTelemetry exporter yourself, as described in the section called “Send telemetry to CloudWatch Omni”.
- The AgentCore CLI, if your agent runs on AgentCore: `npm install -g @aws/agentcore` (requires Node.js 20 or later).
- Model access for two models: the one your agent calls, and the judge model your custom evaluator uses in steps 11 and 14. You choose the judge model when you create the evaluator. For the sample agent both are Amazon Bedrock models, so model access means the models are enabled in your account.
- Credentials the extension can use to call your agent's model while the agent runs on your machine in steps 2 and 3. This is separate from model access: it is how the extension authenticates. The sample agent calls Bedrock, so for it that means AWS credentials, and step 1 covers where they go.

Choose your agent

The sample agent is AZ Customer Support, a multi-agent e-commerce agent, and it produces the scope violation this tutorial fixes. If you have your own agent, use it. The method is the same, and this table is what changes.

Step	Sample agent	Your own agent
1	Choose **Use Sample Project**.	Choose **Open Folder** and open your project.

Step	Sample agent	Your own agent
2	Ships uninstrumented, so onboarding instruments it.	If it already sends OpenTelemetry traces, verify them instead of instrumenting again.
3	Send the four requests given in the step.	Send requests your agent handles, covering its tools if it has them.
4	Deploy with `agentcore deploy`.	Deploy however you deploy today, and configure the exporter yourself.
5	Ask the off-topic questions given. The agent answers them.	Send the failing request in a few variations. See below.
11	Score against AZ Customer Support's permitted topics.	Score against your agent's rule, written the same way.
12	Append the scope restriction given in the step.	Append the equivalent instruction in your agent's own terms.
13	Redeploy and re-invoke with the AgentCore CLI.	Redeploy and re-invoke the same way you did in steps 4 and 5.

Steps not in the table are the same either way.

Choosing your failing scenario is the one substitution that takes judgment, and it is worth settling before you start, because the rest of the tutorial depends on all three of these being true:

- **It completes successfully** — no exception, no error status. A failure that already surfaces as an error does not need this workflow.

- **General-purpose evaluators miss it** — helpfulness or correctness score it acceptably, because the response is well-formed and merely wrong for your agent.
- **The rule fits in one sentence** — "it must not answer questions outside returns, products, and orders" can be scored consistently; "it should be more helpful" cannot.

Answering an out-of-scope request is a common failure that qualifies, which is why the sample uses it. Calling the wrong tool, inventing an identifier, and skipping a required disclaimer also qualify.

Phase A: Build and instrument (IDE extension)

Steps 1–3 happen in your editor.

Step 1: Set up the extension and open your agent project

1. Install the CloudWatch Omni extension in your editor. Search for it in your editor's extensions view, or install it from the registry your editor reads: Install for VS Code from the VS Code Marketplace, or Install for Kiro and Cursor from Open VSX.
2. Open the Omni panel, then get a project:
   - **Sample agent:** choose **Use Sample Project** and pick a destination folder. The sample is copied there and opens automatically.
   - **Your own agent:** choose **Open Folder** and open its project.
3. Stay in local mode, the default until you connect to a space in step 9. Traces and results stay on your machine while you develop.
4. Give the extension credentials for the model your agent calls, so that it can run. In the login panel's **Local dev environment** card, connect an AWS account if your agent calls Amazon Bedrock (as the sample agent does), or add a provider key there for a non-Bedrock model. If you would rather sign in to your space first, its AWS account can cover local development instead; step 9 covers that choice.

The extension creates an `.omni` folder in your project for local traces and configuration, and keeps it out of source control. For the local and cloud mode model, see the section called “Develop with the IDE extension”.

Checkpoint: the extension's panel is open on your project and reports local mode.

Step 2: Instrument your agent

Instrumentation decides what your traces contain. An evaluator cannot score content that was never recorded.

1. Start the onboarding workflow. AI-assisted instrumentation detects your framework and language, checks for existing or conflicting telemetry, installs the dependencies, and adds the configuration. Review the changes it proposes before applying them.
2. To instrument manually instead, start from your framework's entry in the section called "Supported environments, languages, and frameworks" and follow the procedure in the section called "Send AI agent telemetry". The per-framework path is what produces a properly nested trace instead of a flat list of model calls.

If your own agent already sends OpenTelemetry traces, do not instrument it a second time: two instrumentation approaches in one process produce duplicate spans, empty attributes, or no traces at all. Go to step 3 and check what your existing traces contain; if input and output are missing, the section called "Send AI agent telemetry" covers the settings that control content capture.

Checkpoint: onboarding reports instrumentation verified.

Step 3: Generate your first traces locally

1. Go to **Test Agent** in the extension and start the local dev server.
2. Send the agent a few requests. If it has tools, cover more than one of them. For the sample agent, send each of these in turn:

```
What's your return policy?
```

```
I want to return the clothing item on order 404
```

```
Do you have wireless headphones in stock?
```

```
What's my loyalty points balance?
```

   For your own agent, send requests it handles today. The point is to produce a few traces you can inspect, not to reproduce anything yet.

3. Open **Agent traces** in the extension and inspect a trace.

Checkpoint: the trace is nested (agent and model spans, plus tool spans if your agent uses tools) and it shows input and output content. A flat or blank trace means instrumentation is wrong. Fix it before continuing, starting from the symptom table in the section called "Send AI agent telemetry". Every later step depends on correct instrumentation.

Phase B: Deploy and observe (Omni web UI)

Steps 4–8 happen in the Omni web UI: open your space in CloudWatch Omni, in the account and Region you set it up in. You return to the extension in step 9.

Step 4: Deploy the agent and send telemetry to CloudWatch

1. For an agent hosted on Amazon Bedrock AgentCore, enable Transaction Search once per account, then allow about 10 minutes:

   ```
   aws xray update-trace-segment-destination --destination CloudWatchLogs
   ```

2. Make prompt and response content visible in traces: evaluators cannot score content they cannot see. With the AgentCore CLI, set the content environment variable in your agent's `envVars` in `agentcore.json`:

   ```
   "envVars": [{ "name": "AWS_GENAI_CONTENT_EXTRACTION_OPT_OUT", "value": "true" }]
   ```

   This is a one-time edit: runtime environment variables are part of the project configuration and apply on every deploy. For what this variable does and the others that govern content capture, see the section called "Send AI agent telemetry".

3. Deploy the agent so that the setting takes effect and telemetry flows:

   ```
   agentcore deploy
   ```

   The AgentCore runtime turns on agent observability for you.

   If your agent runs anywhere else — Lambda, Amazon ECS, Amazon EKS, your own hosts — deploy it the way you already do, and configure the OpenTelemetry exporter and its permissions yourself. the section called "Send telemetry to CloudWatch Omni" has the endpoint, the

authentication, and the same content setting for your runtime. The rest of this tutorial does not depend on how you deployed, only on traces arriving in your space.

Checkpoint: The agent is deployed, and its traces appear in **Agent traces** in your space. Allow about 10 minutes from enabling Transaction Search.

Step 5: Drive traffic against the deployed agent

Send the agent a mix of normal requests and more than one request you expect it to mishandle. Step 8 packages the failing runs into a dataset, and a single run gives it only one test case. For the sample agent:

```
agentcore invoke "What is your return policy for clothing items?"
```

```
agentcore invoke "Hi, can you give me a quick chocolate chip cookie recipe?"
```

```
agentcore invoke "What's a good stretch for lower back pain?"
```

```
agentcore invoke "Process a return for order 999"
```

The return-policy request produces a clean trace. The cookie-recipe and stretching requests are the problem this tutorial detects and fixes: the agent answers both. The order-return request produces a multi-tool trace.

For your own agent, call it the way you normally do — its own CLI, an HTTP request, or your test client — and send the failing request you chose in **Choose your agent** in two or three variations, plus at least one request it handles correctly. You need both kinds: the clean run is what you compare the failing one against in step 7 and what proves in step 11 that your evaluator is not failing everything, and the failing variations become the dataset in step 8.

Checkpoint: all your requests are visible in **Agent traces**, allowing for the ingestion delay from step 4, and the ones you expected to fail did fail. If they did not, try requests further outside what your agent should handle before going on. The rest of the tutorial needs real failing runs, not hypothetical ones.

Step 6: Monitor token usage and cost

These are the views an operator opens first. Work through them before you open any single run, partly to learn them and partly because of what they will not tell you.

1. Open **Agent Dashboards** to see all your agents in one view, and review request volume, errors, latency, and token usage across them. Token usage is the signal that drives cost. For what each signal tells you, see the section called “Monitor agents in the web UI”.
2. Filter to a single agent to see the same signals for only that agent, and compare its token consumption to the rest.
3. When a trend needs explaining, use the section called “Agent topology” to see where latency and errors concentrate across your agent's components.

Now notice what is absent: the failure you are hunting does not appear anywhere in this step. Volume, errors, and latency all look healthy, because the agent answered promptly and successfully. It answered the wrong question. Performance monitoring cannot see that, which is the first half of why response quality needs measuring on its own. Step 7 is the second half.

Checkpoint: you can answer "which agent consumes the most tokens, and is that changing?" You have also confirmed that nothing here flags the failing run.

Step 7: Find the anomalous trace

A failure like this raises no alarm. The run succeeded, so nothing routed you to it. You find it by opening the run and reading what the agent actually did. This step also produces the evidence that makes the next phase necessary.

1. In **Agent traces**, open the failing run (for the sample agent, the cookie-recipe request). For multi-turn context, use **Agent sessions**. See the section called “Agent traces” and the section called “Agent sessions”.
2. Select that trace and the closest well-handled one (for the sample, the return-policy run) and compare them side by side. The clean run calls tools to look up policy; the failing run calls none, so nothing external handed the agent a recipe. The cause is the system prompt, which is what you change in step 12. See the section called “Agent traces” and the section called “Agent sessions”.
3. Run a built-in evaluator on the failing run, such as one that scores helpfulness.

Why the failing run is wrong. The sample's response is fluent, friendly, and a perfectly good recipe. That is the problem: cooking questions are outside an AZ Customer Support agent's scope. It spends tokens on work that is not its job, and it puts the company behind advice it never meant to give. Nothing about the run is technically broken, which is exactly why error rates, latency, and throughput never surfaced it.

Checkpoint: you have a failing run you can explain, a clean run to compare it against, and a built-in evaluator score showing that existing checks rate the failing run as acceptable.

Step 8: Package the failing runs into a dataset

1. In **Agent traces**, filter to the failing traces and select them: for the sample agent, the cookie-recipe and stretching runs; for your own, the failing variations from step 5.
2. Add them to a new dataset named `scope-regression`. For your own agent, name it after the failure you are fixing. Later steps say `scope-regression`, so substitute your name as you go.
3. Publish the dataset as a numbered version, which is what later playground runs and experiments read.

The dataset is the handoff artifact. It carries the real failing inputs, so the engineer reproduces the actual problem instead of a description of it. See the section called “Datasets”.

Checkpoint: `scope-regression` exists in your space with the failing test cases.

Phase C: Reproduce and fix (IDE extension)

Steps 9–13 happen back in your editor, and they move the extension through both of its modes. In **local mode**, the extension reads traces and datasets from your machine. In **cloud mode**, it reads them from your space. Playground runs, evaluators, and experiments always execute on your machine, in either mode.

Step 9 connects to your space, which switches the extension to cloud mode. Step 10 downloads the dataset from the space. Step 12 switches back to local mode to run the fix. The status bar shows which mode is active throughout.

Step 9: Connect the extension to your space

Up to now the extension has been in local mode, working only against traces on your machine. Connecting it to your space puts it on the same data as the Omni web UI, which is what lets you import the dataset in step 10.

The extension has two independent sign-ins, and this step is about the first one:

- **Omni Space** gives you cloud traces, datasets, and evaluations from the space you created for this tutorial.
- **Local dev environment** provides the AWS account that runs the prompt playground and evaluators on your machine.
- Open the extension's **Login** panel. The **Omni Space** card shows whether you are connected.
- Choose **Sign In**. Your browser opens your domain's Omni sign-in page, the same page and the same identity you use to reach the Omni web UI. Complete sign-in there.
- The browser hands you back to your editor, which confirms that you are signed in. Leave the browser tab open until it does.
- Choose your space. If your domain has one space, the extension selects it for you. If it has several, a picker opens. Select the space you created.
- Cloud mode turns on by itself once a space is selected. The status bar shows which mode is active, and it is also how you switch back to local mode in step 12.
- Leave **Use this account for local dev** selected. Local dev then runs on your space's AWS account, so steps 11 and 12 need no second set of credentials. Clear it only if you want the playground and evaluators to run against a different AWS account, which you then connect in the **Local dev environment** card.

Checkpoint: the **Omni Space** card shows **Connected**, with your space name and domain, and the panel headers show a **Cloud** badge instead of **Local**.

> **Note**
>
> Your cloud session expires, and a long tutorial can outlive it. When it does, the extension tells you and offers to sign in again, and panels show an expired-session banner with a sign-in action. Signing in again returns you to the step you were on. Nothing you created in the space is lost.

For what local and cloud mode each cover, see the section called “Develop with the IDE extension”.

Step 10: Import the dataset created in the Omni web UI

This step runs in cloud mode, which step 9 turned on: the **Datasets** panel lists the datasets in your space rather than the ones on your machine.

1. Open **Datasets** in the extension. `scope-regression` appears in the list.
2. Select it, and then choose **Download to Local**.

The download creates an independent local copy of the dataset. There is no file to export and nothing to re-upload. The copy carries its own version history and does not sync back, so later changes to one do not reach the other. See the section called “Datasets”.

Checkpoint: the download completes, and `scope-regression` exists as a local dataset. You see it listed when you switch to local mode in step 12.

Step 11: Create a custom evaluator that detects the issue

The failure is a judgment about meaning, so create a custom model-as-judge evaluator. (A code-based evaluator suits mechanical checks such as valid JSON or length limits, not this.) For how to specify a judge (stating scope, writing decidable cases, and matching the rating scale to the criterion), see the section called “Evaluators and evaluations”.

1. In **Evaluators**, create a new evaluator, choose model-as-judge, and name it `On-Topic Compliance`. For your own agent, name it after the rule it enforces. Later steps say `On-Topic Compliance`.
2. Write the rubric. For the sample agent:

```
You are evaluating whether a customer support agent stayed within its
designated scope.

The agent is an AZ Customer Support e-commerce agent. Its ONLY
permitted topics are: returns, refunds, and exchanges; product inquiries;
and account and order management.

Rubric:
- On-Topic (score 1): the response addresses only permitted topics, or
  politely declines an off-topic request.
- Off-Topic (score 0): the response answers anything outside the permitted
  topics (recipes, general knowledge, weather, coding help, trivia).
  A response that acknowledges the request is off-topic but still answers
```

```
  it is Off-Topic.

## Conversation context
{context}

## Response under evaluation
{assistant_turn}
```

Omni fills the template placeholders with the conversation context and the response under evaluation.

For your own agent, keep this shape and replace the specifics: say what the agent is, enumerate what it may do, then define each score with the conditions that earn it (including the edge case that trips judges most often, a response that acknowledges the rule and breaks it anyway).

3. Make the rating scale binary: score `0` for the failure and score 1 for correct behavior, each with a one-line definition, and remove any middle values.
4. Test it on a known-bad and a known-good trace. The failing trace must score 0 and the clean one must score 1. If either misses, tighten the rubric.

Checkpoint: the evaluator fails the failing trace and passes the clean one. Both halves matter: an evaluator that fails everything looks like it works until it blocks your fix.

Step 12: Compare a fix against the baseline

1. Switch the extension back to local mode. Choose the mode indicator in the status bar, and then choose **Switch to Local**. The dataset copy from step 10 is a local dataset, and the experiments you run against it are available only in local mode.
2. If your prompts are not yet under management, set up prompt management so that the system prompt is a versioned artifact. The sample agent ships with managed prompts, and its system prompt is `MAIN_AGENT`. See the section called “Manage prompts”.
3. Open the **Prompt playground** and choose a mode. **LLM** tests a prompt in isolation. **Agent** runs your full agent against the dev server with the draft prompt applied. Choose **LLM** for this step, because the change under comparison is a system prompt change.
4. Set up two variants: a baseline with the current prompt, unchanged, and a candidate carrying your fix. Both variants run the same inputs, so any difference in their scores can only come from your change. Edit both and you can no longer say which edit fixed anything. See the section called “Prompt playground”.

5. On the candidate, append a scope restriction to the system prompt. For the sample agent:

```
SCOPE RESTRICTION: You may only assist with AZ Customer Support
topics: returns, refunds, exchanges, product inquiries, and account and
order management. If a customer asks about anything else - recipes, coding
help, general knowledge, weather, trivia - politely decline and offer to
help with a supported topic instead. Do not answer the off-topic question
even partially.
```

   For your own agent, write the same instruction in its terms: state what it may do, say what to do with anything else, and rule out answering partially.

6. Run both variants against the imported `scope-regression` dataset and score both with `On-Topic Compliance`. The baseline should fail the way production fails, which confirms you reproduced the problem, and the candidate should pass.
7. Iterate on the candidate until it passes across the dataset. For repeatable version-against-version comparisons, see the section called “Experiments”.

If your fix is not a prompt change — a tool that needs rewiring, a retrieval step returning the wrong context, orchestration taking the wrong branch — the loop is the same. Choose **Agent** in the playground rather than **LLM**: it runs your agent end to end, tools and orchestration included, against the same dataset and the same evaluator. Run it as an experiment when you want the comparison recorded and repeatable rather than interactive. See the section called “Prompt playground” and the section called “Experiments”.

Checkpoint: the baseline scores 0 on the failing cases, and the candidate scores 1 on all of them.

Step 13: Save the prompt version and redeploy

1. Save the winning prompt as a new version, with a note describing the fix. See the section called “Manage prompts”.
2. Redeploy the agent with the saved version. For an agent on AgentCore:

```
agentcore deploy
```

   For any other runtime, deploy the way you did in step 4.

3. Send traffic that covers the failing scenario (one request the agent should handle and one it previously got wrong) so that fresh traces flow to your space. For the sample agent:

```
agentcore invoke "What is your return policy?"
```

```
agentcore invoke "Give me a cookie recipe"
```

For your own agent, send the same pair you used in step 5, the same way you sent them.

Checkpoint: New traces from the redeployed agent appear in your space, and the request that used to be mishandled now produces the corrected behavior (for the sample agent, a polite refusal instead of a recipe). Compare the response, not the tool calls: a declined request calls no tools, so the trace's tool activity looks the same as it did before the fix.

Phase D: Verify continuously (Omni web UI)

Step 14 happens in the Omni web UI, in the same space.

Step 14: Watch the fix on live traffic

An online evaluation scores sampled live traffic continuously, so that this regression cannot return unnoticed. See the section called “Online evaluations”.

1. In your space, choose **Online evaluations**, and then create a new evaluation.
2. Select the agent to evaluate, select the `On-Topic Compliance` evaluator, and set the sampling percentage. Sampling is your cost control: the judge model is invoked for every sampled trace, so evaluate a fraction of high-volume traffic.
3. Grant the permissions the evaluation needs to write its results. The Omni web UI guides you through this step.
4. Enable the evaluation and confirm that its status shows enabled. To change its evaluators later, disable it first: the evaluator list is locked while it runs.
5. In the **Evaluation dashboard**, confirm that the pass rate recovers on the newest traces.

Checkpoint: the online evaluation is enabled, and `On-Topic Compliance` trends at 1 on live traffic.

What you keep

Each artifact in the loop does one job, and each keeps working after the fix ships.

- **The dataset** `scope-regression` pins down the exact failing scenarios. It is now a regression test: run it in an experiment whenever you change the agent. See the section called “Datasets”.
- **The evaluator** `On-Topic Compliance` defines what "broken" means identically wherever it runs: in the playground during the fix, and on live traffic afterward.
- **The prompt version** records what changed and when. See the section called “Manage prompts”.

Because both surfaces share one space, the operator's evidence became the engineer's test: no exports, screenshots, or re-described bugs in between.

The next quality problem will not be a scope violation, but the shape is the same: name the failure precisely enough to score it, collect the runs that fail it, fix against an unchanged baseline, and leave the evaluator running.

Using Amazon CloudWatch dashboards

Amazon CloudWatch includes automatic pre-built dashboards and also enables you to create your own dashboards. Dashboards help you to monitor your resources in a single view, even those resources that are spread across different Regions. You can use CloudWatch dashboards to create customized views of the telemetry data for your AWS resources.

With customizable dashboards, you can create the following:

- A single view for selected metrics and alarms to help you assess the health of your resources and applications across one or more Regions. You can select the color used for each metric on each graph, so that you can easily track the same metric across multiple graphs.
- An operational playbook that provides guidance for team members during operational events about how to respond to specific incidents.
- A common view of critical resource and application measurements that can be shared by team members for faster communication flow during operational events.

If you have multiple AWS accounts, you can set up *CloudWatch cross-account observability* and then create rich cross-account dashboards in your monitoring accounts. You can seamlessly search, visualize, and analyze your metrics, logs, and traces without account boundaries.

With CloudWatch cross-account observability, you can do the following in a dashboard in a monitoring account:

- Search, view, and create graphs of metrics that reside in source accounts. A single graph can include metrics from multiple accounts.
- Create alarms in the monitoring account that watch metrics in source accounts.
- View the log events from log groups located in source accounts, and run CloudWatch Logs Insights queries of log groups in source accounts. A single CloudWatch Logs Insights query in a monitoring account can query multiple log groups in multiple source accounts at once.
- View nodes from source accounts in a trace map in X-Ray. You can then filter the map to specific source accounts.

When you are signed in to a monitoring account, a blue **Monitoring account** badge appears at the top right of every page that supports CloudWatch cross-account observability functionality.

For more information about setting up CloudWatch cross-account observability, see CloudWatch cross-account observability.

You can create dashboards from the console or using the AWS CLI or `PutDashboard` API operation. You can add dashboards to a favorites list, where you can access not only your favorite dashboards, but also your recently visited dashboards. For more information, see Add a dashboard to your favorites list.

To access CloudWatch dashboards, you need one of the following:

- The `AdministratorAccess` policy
- The `CloudWatchFullAccess` policy
- A custom policy that includes one or more of these specific permissions:
  - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards
  - `cloudwatch:PutDashboard` to be able to create or modify dashboards
  - `cloudwatch:DeleteDashboards` to be able to delete dashboards

Topics

- Getting started with CloudWatch automatic dashboards
- Creating a customized CloudWatch dashboard
- Creating a CloudWatch cross-account cross-Region dashboard with the AWS Management Console
- Create a cross-account cross-Region dashboard programmatically
- Creating a graph with metrics from different accounts and Regions in a CloudWatch dashboard
- Adding an alarm from a different account to a CloudWatch cross-account dashboard
- Creating flexible CloudWatch dashboards with dashboard variables
- Using widgets on CloudWatch dashboards
- Sharing CloudWatch dashboards
- Using live data in CloudWatch dashboards
- Viewing an animated CloudWatch dashboard
- Adding a CloudWatch dashboard to your favorites list
- Changing the period override setting or refresh interval for the CloudWatch dashboard
- Changing the time range or time zone format of a CloudWatch dashboard

- Dashboard Body Structure and Syntax
- Graphing metrics
- Explore related telemetry

Getting started with CloudWatch automatic dashboards

The CloudWatch home page automatically displays metrics about every AWS service you use. You can additionally create custom dashboards to display metrics about your custom applications, and display custom collections of metrics that you choose.

Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

The CloudWatch overview home page appears.



The overview displays the following items, refreshed automatically.

- **Alarms by AWS service** displays a list of AWS services you use in your account, along with the state of alarms in those services. Next to that, two or four alarms in your account are displayed. The number depends on how many AWS services you use. The alarms shown are those in the ALARM state or those that most recently changed state.

  These upper areas help you quickly assess the health of your AWS services, by seeing the alarm states in every service and the alarms that most recently changed state. This helps you monitor and quickly diagnose issues.

- Below these areas is the *default dashboard*, if one exists. The default dashboard is a custom dashboard that you have created and named **CloudWatch-Default**. This is a convenient way for you to add metrics about your own custom services or applications to the overview page, or to bring forward additional key metrics from AWS services that you most want to monitor.

> **Note**
>
> The automatic dashboards on the CloudWatch home page display only information from the current account, even if the account is a monitoring account set up for CloudWatch cross-account observability. For information about creating custom cross-account dashboards, see Creating a CloudWatch cross-account cross-Region dashboard with the AWS Management Console.

From this overview, you can see a cross-service dashboard of metrics from multiple AWS service, or focus your view to a specific resource group or a specific AWS service. This enables you to narrow your view to a subset of resources in which you are interested.

Topics

- Viewing the cross-service CloudWatch dashboard
- Removing a service from appearing in the CloudWatch cross-service dashboard
- Viewing a CloudWatch dashboard for a single AWS service
- Viewing a CloudWatch dashboard for a resource group

Viewing the cross-service CloudWatch dashboard

You can switch to the Cross-service dashboard screen and interact with dashboards for all of the AWS services that you're using. The CloudWatch Console displays your dashboards in alphabetical order and displays one or two key metrics from each service.

> **Note**
>
> If you're using five or more AWS services, the CloudWatch Console won't display the Cross-service dashboard on the Overview screen.

To view the Cross-service dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   You're directed to the Overview screen.

2. From the Overview screen, select the dropdown that reads **Overview**, and then choose **Cross service dashboard**.

   You're directed to the Cross service dashboard screen.

3. (Optional) If you're using the original interface, scroll to the section **Cross-service dashboard**, and then choose **View Cross-service dashboard**.

   You're directed to the Cross-service dashboard screen.

4. You can focus on a particular service in two ways:

   a. To see more key metrics for a service, choose its name from the list at the top of the screen, where **Cross service dashboard** is currently shown. Or, you can choose **View *Service* dashboard** next to the service name.

      An automatic dashboard for that service is displayed, showing more metrics for that service. Additionally, for some services, the bottom of the service dashboard displays resources related to that service. You can choose one of those resources to that service console and focus further on that resource.

   b. To see all the alarms related to a service, choose the button on the right of the screen next to that service name. The text on this button indicates how many alarms you have created in this service, and whether any are in the ALARM state.

      When the alarms are displayed, multiple alarms that have similar settings (such as dimensions, threshold, or period) may be shown in a single graph.

      You can then view details about an alarm and see the alarm history. To do so, hover on the alarm graph, and choose the actions icon, **View in alarms**.

      The alarms view appears in a new browser tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.

5. You can focus on resources in a particular resource group. To do so, choose the resource group from the list at the top of the page where **All resources** is displayed.

   For more information, see Viewing a CloudWatch dashboard for a resource group.

6. To change the time range shown in all graphs and alarms currently displayed, select the range you want next to **Time range** at the top of the screen. Choose **custom** to select from more time range options than those displayed by default.
7. Alarms are always refreshed once a minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose the refresh rate you want. You can also choose to turn off automatic refresh.

Removing a service from appearing in the CloudWatch cross-service dashboard

You can prevent a service's metrics from appearing in the cross-service dashboard. This helps you focus your cross-service dashboard on the services you most want to monitor.

If you remove a service from the cross-service dashboard, the alarms for that service still appear in the views of your alarms.

To remove a service's metrics from the cross-service dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   The home page appears.
2. At the top of the page, under **Overview**, choose the service you want to remove.

   The view changes to show metrics from only that service.
3. Choose **Actions**, then clear the check box next to **Show on cross service dashboard**.

Viewing a CloudWatch dashboard for a single AWS service

On the CloudWatch home page, you can focus the view to a single AWS service. You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows only how to focus on an AWS service.

To focus on a single service

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   The home page appears.

2. For **Overview**, where **Overview** is currently shown in the drop down menu, choose **Service dashboards**.
3. Choose the service that you want to focus on.

   The view changes to display graphs of key metrics from the selected service.
4. To switch to viewing the alarms for this service, select the check box for **In alarm**, **Insufficient data**, or **OK** near the top of the screen where the service name is currently displayed.
5. When viewing metrics, you can focus on a particular metric in several ways:

   a. To see more details about the metrics in any graph, hover on the graph, and choose the actions icon, **View in metrics**.

      The graph appears in a new tab, with the relevant metrics listed below the graph. You can customize your view of this graph, changing the metrics and resources shown, the statistic, the period, and other factors to get a better understanding of the current situation.

   b. You can view log events from the time range shown in the graph. This may help you discover events that happened in your infrastructure that are causing an unexpected change in your metrics.

      To see the log events, hover on the graph, and choose the actions icon, **View in logs**.

      The CloudWatch Logs view appears in a new tab, displaying a list of your log groups. To see the log events in one of these log groups that occurred during the time range shown in the original graph, choose that log group.
6. When viewing alarms, you can focus on a particular alarm in several ways:

   - To see more details about an alarm, hover on the alarm, and choose the actions icon, **View in alarms**.

     The alarms view appears in a new tab, displaying a list of your alarms, along with details about the chosen alarm. To see the history for this alarm, choose the **History** tab.
7. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.

8. To change the time range shown in all graphs and alarms currently displayed, next to **Time range** at the top of the screen, choose the range. To select from more time range options than those displayed by default, choose **custom**.
9. To return to the cross-service dashboard, choose **Overview** in the list at the top of the screen that currently shows the service you are focusing on.

   Alternatively, from any view, you can choose **CloudWatch** at the top of the screen to clear all filters and return to the overview page.

Viewing a CloudWatch dashboard for a resource group

You can focus your view to display metrics and alarms from a single resource group. Using resource groups enables you to use tags to organize projects, focus on a subset of your architecture, or distinguish between your production and development environments. They also enable you to focus on each of these resource groups on the CloudWatch overview. For more information, see What Is AWS Resource Groups?.

When you focus on a resource group, the display changes to show only the services where you have tagged resources as part of this resource group. The recent alarms area shows only the alarms that are associated with resources that are part of the resource group. Additionally, if you have created a dashboard with the name **CloudWatch-Default-*ResourceGroupName***, it is displayed in the **Default dashboard** area.

You can drill down further by focusing on both a single AWS service and a resource group at the same time. The following procedure shows just how to focus on a resource group.

To focus on a single resource group

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. At the top of the page, where **All resources** is displayed, choose a resource group.
3. To see more metrics related to this resource group, near the bottom of the screen, choose **View cross service dashboard**.

   The cross-service dashboard appears, showing only the services related to this resource group. For each service, one or two key metrics are displayed.
4. To change the time range shown in all graphs and alarms currently displayed, for **Time range** at the top of the screen, select a range. To select from more time range options than those displayed by default, choose **custom**.

5. Alarms are always refreshed one time per minute. To refresh the view, choose the refresh icon (two curved arrows) at the top right of the screen. To change the automatic refresh rate for items on the screen other than alarms, choose the down arrow next to the refresh icon and choose a refresh rate. You can also choose to turn off automatic refresh.
6. To return to showing information about all the resources in your account, near the top of the screen where the name of the resource group is currently displayed, choose **All resources**.

Creating a customized CloudWatch dashboard

To get started, create a CloudWatch dashboard. You can create multiple dashboards, and you can add dashboards to a favorites list. You aren't limited to the number of dashboards that you can have in your AWS account. All dashboards are global. They are not Region-specific.

> **Opt-in Region propagation delay**
>
> You might have recently enabled an opt-in Region (Region that is disabled by default) for your account. If so, your existing dashboards can take up to 24 hours to be available in that Region. During this period, `GetDashboard` and `ListDashboards` requests might return an access denied error. This does not indicate a problem with your permissions. Because dashboards are global, you can view them from any other enabled Region while propagation completes. Creating and updating dashboards in the new Region works immediately.

The following procedure shows you how to create a dashboard from the CloudWatch console. You can use the `PutDashboard` API operation to create a dashboard from the command line interface. The API operation contains a JSON string that defines your dashboard content. For more information about creating a dashboard with the `PutDashboard` API operation, see PutDashboard in the *Amazon CloudWatch API Reference*.

> **Tip**
>
> If you're creating a new dashboard with the `PutDashboard` API operation, you can use the JSON string from a dashboard that already exists.

To create a dashboard from the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose **Create dashboard**.
3. In the **Create new dashboard** dialog box, enter a name for the dashboard, and then choose **Create dashboard**.

   If you use the name **CloudWatch-Default** or **CloudWatch-Default-*ResourceGroupName***, the dashboard appears in the overview of the CloudWatch home page under **Default Dashboard**. For more information, see Getting started with CloudWatch automatic dashboards.
4. In the **Add to this dashboard** dialog box, do one of the following:
   - To add a graph to the dashboard, choose **Line** or **Stacked area**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**. If a metric doesn't appear in the dialog box because it hasn't published data in more than 14 days, you can add it manually. For more information, see Graph metrics manually on a CloudWatch dashboard.
   - To add a number displaying a metric to the dashboard, choose **Number**, and then choose **Configure**. In the **Add metric graph** dialog box, select the metric(s) to graph, and then choose **Create widget**.
   - To add a text block to the dashboard, choose **Text**, and then choose **Configure**. In the **New text widget** dialog box, for **Markdown**, format your text using Markdown, and then choose **Create widget**.
5. (Optional) Choose **Add widget**, and then repeat step 4 to add another widget to the dashboard. You can repeat this step multiple times.

   For each graph on the dashboard, there is an information icon at the upper right. Choose this icon to see the descriptions of the metrics in the graph.
6. Choose **Save dashboard**.

Creating a CloudWatch cross-account cross-Region dashboard with the AWS Management Console

You can create *cross-account cross-Region dashboards*, which summarize your CloudWatch data from multiple AWS accounts and multiple Regions into one dashboard. From this high-level

dashboard you can get a view of your entire application, and also drill down into more specific dashboards without having to sign in and out of accounts or switch Regions.

Prerequisite

Before you can create a cross-account cross-Region dashboard, you must enable at least one sharing account and at least one monitoring account. Additionally, to be able to use the CloudWatch console to create a cross-account dashboard, you must enable the console for cross-account functionality. For more information, see Cross-account cross-Region CloudWatch console.

To create a cross-account cross-Region dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the navigation pane, choose **Dashboards**.
4. Choose a dashboard, or create a new one.
5. At the top of the screen, you can switch between accounts and Regions. As you create your dashboard, you can include widgets from multiple accounts and Regions. Widgets include graphs, alarms, and CloudWatch Logs Insights widgets.

Create a cross-account cross-Region dashboard programmatically

You can use the AWS APIs and SDKs to create dashboards programmatically. For more information, see PutDashboard.

To enable cross-account cross-Region dashboards, we have added new parameters to the dashboard body structure, as shown in the following table and examples. For more information about overall dashboard body structure, see Dashboard Body Structure and Syntax.

Parameter	Use	Scope	Default
`accountId`	Specifies the ID of the account where the widget or the metric is located.	Widget or metric	Account that is currently logged in

Parameter	Use	Scope	Default
`region`	Specifies the Region of the metric.	Widget or metric	Current Region selected in the console

The following examples illustrate the JSON source for widgets in a cross-account cross-Region dashboard.

This example sets the `accountId` field to the ID of the sharing account at the widget level. This specifies that all metrics in this widget will come from that sharing account and Region.

```
{
  "widgets": [
    {
          ...
          "properties": {
        "metrics": [
                  …
        ],
        "accountId": "111122223333",
        "region": "us-east-1"
      }
    }
  ]
}
```

This example sets the `accountId` field differently at the level of each metric. In this example, the different metrics in this metric math expression come from different sharing accounts and different Regions.

```
{
  "widgets": [
    {
          ...
          "properties": {
```

```
        "metrics": [
          [
            {
              "expression": "SUM(METRICS())",
              "label": "[avg: ${AVG}] Expression1",
              "id": "e1",
              "stat": "Sum"
            }
          ],
          [
            "AWS/EC2",
            "CPUUtilization",
            {
              "id": "m2",
              "accountId": "5555666677778888",
              "region": "us-east-1",
              "label": "[avg: ${AVG}] ApplicationALabel "
            }
          ],
          [
            ".",
            ".",
            {
              "id": "m1",
              "accountId": "9999000011112222",
              "region": "eu-west-1",
              "label": "[avg: ${AVG}] ApplicationBLabel"
            }
          ]
        ],
        "view": "timeSeries",
        "region": "us-east-1", ---> home region of the metric. Not present in above
 example
              "stacked": false,
        "stat": "Sum",
        "period": 300,
        "title": "Cross account example"
      }
    }
  ]
}
```

This example shows an alarm widget.

```
{
  "type": "metric",
  "x": 6,
  "y": 0,
  "width": 6,
  "height": 6,
  "properties": {
    "accountID": "111122223333",
    "title": "over50",
    "annotations": {
      "alarms": [
        "arn:aws:cloudwatch:us-east-1:379642911888:alarm:over50"
      ]
    },
    "view": "timeSeries",
    "stacked": false
  }
}
```

This example is for a CloudWatch Logs Insights widget.

```
{
  "type": "log",
  "x": 0,
  "y": 6,
  "width": 24,
  "height": 6,
  "properties": {
    "query": "SOURCE 'route53test' | fields @timestamp, @message\n| sort @timestamp
 desc\n| limit 20",
    "accountId": "111122223333",
    "region": "us-east-1",
    "stacked": false,
    "view": "table"
  }
}
```

Another way to create dashboards programmatically is to first create one in the AWS Management Console, and then copy the JSON source of this dashboard. To do so, load the dashboard and choose **Actions**, **View/edit source**. You can then copy this dashboard JSON to use as a template to create similar dashboards.

Creating a graph with metrics from different accounts and Regions in a CloudWatch dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**.
4. Select the account and Region that you want to add metrics from. You can select your account and Region from the account and Region dropdown menus near the top-right of the screen.
5. Add the metrics you want to the graph. For more information, see Graphing metrics.
6. Repeat steps 4-5 to add metrics from other accounts and Regions.
7. (Optional) Choose the **Graphed metrics** tab and add a metric math function that uses the metrics that you have chosen. For more information, see Math expressions with metrics.

   You can also set up a single graph to include multiple SEARCH functions. Each search can refer to a different account or Region.
8. When you are finished with the graph, choose **Actions**, **Add to dashboard**.

   Select your cross-account dashboard, and choose **Add to dashboard**.

Adding an alarm from a different account to a CloudWatch cross-account dashboard

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. At the top of the page, choose the account where the alarm is located.
4. In the navigation pane, choose **Alarms**.
5. Select the check box next to the alarm that you want to add, and choose **Add to dashboard**.

6. Select the cross-account dashboard that you want to add it to, and choose **Add to dashboard**.

Creating flexible CloudWatch dashboards with dashboard variables

Use *dashboard variables* to create flexible dashboards that can quickly display different content in multiple widgets, depending on the value of an input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different AWS Regions.

After you create a dashboard that uses a variable, you can copy the same variable pattern to other existing dashboards.

Using dashboard variables improves the operational workflow for people who use your dashboards. It can also reduce your costs because you're using dashboard variables in one dashboard instead of creating multiple similar dashboards.

> **Note**
>
> If you share a dashboard that contains dashboard variables, the people that you share it with won't be able to change between the variable values.

Types of dashboard variables

The dashboard variable can be a *property variable* or a *pattern variable*.

- *Property variables* change all instances of a property in all widgets in the dashboard. This property can be any JSON property in the JSON source of a dashboard, such as `region`. Or it can be a dimension name for a metric, such as `InstanceID` or `FunctionName`.

  For a tutorial that uses a property variable, see Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable.

  For more information about the JSON source of dashboards, see Dashboard Body Structure and Syntax. In the CloudWatch console, you can see the JSON source for any custom dashboard by choosing **Actions**, **View/edit source**.

- *Pattern variables* use a regular expression pattern to change all of a JSON property or only a certain part of it.

  For a tutorial that uses a pattern variable, see Tutorial: Creating a dashboard that uses a regular expression pattern to switch between AWS Regions.

Property variables apply to most use cases and are less complex to set up.

Topics

- Copying a variable to another CloudWatch dashboard
- Tutorial: Creating a dashboard that uses a regular expression pattern to switch between AWS Regions
- Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable

Copying a variable to another CloudWatch dashboard

After you create a dashboard with useful variables, you can copy these variables to other existing dashboards. For more information about dashboard variables, see Creating flexible CloudWatch dashboards with dashboard variables.

To copy a dashboard variable to another dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the name of the dashboard that has the variable that you want to copy. Enter a string to find dashboards with matching names, if needed.
3. Choose **Actions**, **Variables**, **Manage variables**.
4. Choose the radio button next to the variable that you want to copy, and choose **Copy to another dashboard**.
5. Choose the selection box and start typing the dashboard name that you want to copy the variable to.
6. Select the dashboard name and choose **Copy variable**.

Tutorial: Creating a dashboard that uses a regular expression pattern to switch between AWS Regions

The steps in this procedure illustrate how to create a flexible dashboard that can switch between Regions. This tutorial uses a regular expression *pattern variable* instead of a property variable. For a tutorial that uses a property variable, see Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable.

For many use cases, you can create a dashboard that switches between Regions by using a property variable. But if the widgets rely on Amazon Resource Names (ARNs) that include Region names, you must use a pattern variable to change the Region names within the ARNs.

To use a dashboard pattern variable to create a flexible multi-Region dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, **Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard. When you add the widgets that you want to display Region-specific data, avoid specifying any dimensions with values that appear in only one Region. For example, for Amazon EC2 metrics, specify metrics that are aggregated instead of metrics that use **InstanceID** as a dimension.

   For more information about adding widgets to a dashboard, see Using widgets on CloudWatch dashboards.
5. After you add the widgets, as you are viewing the dashboard, choose **Actions**, **Variables**, **Create a variable**.
6. Choose **Pattern variable**.
7. For **What pattern do you want your variable to change?**, enter a Regular Expression (regex) pattern that matches the text you want the variable to replace in your dashboard JSON definition.

   You have the correct pattern entered if the label below the box displays the widgets that will be impacted by the variable.
8. For **Input type**, for this use case, select **Radio button**.
9. For **Define how inputs are populated**, choose **Create a list of custom values**.

10. For **Create your custom values**, enter the Regions that you want to switch between, with one Region on each line. After each Region, enter a comma and then the label to display for that radio button. For example:

    ```
    us-east-1, N. Virginia

    us-east-2, Ohio

    eu-west-3, Paris
    ```

    As you fill in the custom values, the **Preview** pane updates to display what the radio buttons will look like.

11. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:

    - To customize the name of your variable, enter the name in **Custom variable name**.
    - To customize the label for the variable input field, enter the label in **Input label**. For this tutorial, enter **`Region:`**.

      If you enter a value here, the **Preview** pane updates to display what the radio buttons will look like.
    - To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.

12. Choose **Add variable**.

    The dashboard appears, with a **Region:** label next to the radio buttons for the Regions near the top. When you switch between Regions, all the widgets that use the variable will display information about the selected Region.

Tutorial: Creating a CloudWatch Lambda dashboard with function name as the variable

The steps in this procedure illustrate how to create a flexible dashboard that shows a variety of metric graphs, using a property variable. This includes a dropdown selection box on the dashboard that you can use to switch the metrics in all the graphs between different Lambda functions.

Other use case examples for this type of dashboard include using `InstanceId` as the variable to create a dashboard of metrics with a dropdown for instance IDs. Alternatively, you could create

a dashboard that uses `region` as the variable to display the same set of metrics from different Regions.

To use a dashboard property variable to create a flexible Lambda dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, **Create dashboard**.
3. Enter a name for the dashboard, and choose **Create dashboard**.
4. Add widgets to the dashboard that display metrics for a Lambda function. When you create these widgets, specify **Lambda**, **By Function Name** for the widget metrics. For the function, specify one of the Lambda functions that you want to include in this dashboard.

   For more information about adding widgets to a dashboard, see Using widgets on CloudWatch dashboards.
5. After you add the widgets, as you are viewing the dashboard, choose **Actions**, **Variables**, **Create a variable**.
6. Choose **Property variable**.
7. For **Property that the variable changes**, choose **FunctionName**.
8. For **Input type**, for this use case, we recommend choosing **Select menu (dropdown)**. This creates a dropdown menu in the dashboard where you can select the Lambda function name to display metrics for.

   If this was for a dashboard that toggled between only two or three different values for a variable, then **Radio button** would be a good choice.

   If you prefer to enter or paste in values for the variable, you would choose **Text input**. This option doesn't include a dropdown list or radio buttons.
9. When you choose **Select menu (dropdown)**, you must then choose whether to populate the menu by entering values, or using a metric search. For this use case, let's assume that you have a large number of Lambda functions and you don't want to enter all of them manually. Choose **Use the results of a metric search** and then do the following:

   a. Choose **Pre-built queries**, **Lambda**, **Errors**.

      (Choosing **Errors** does not add the **Errors** metric to the dashboard. However, it quickly populates the **FunctionName** variable selection box.)

   b. Choose **By Function Name** and then choose **Search**.

Under the **Search** button, you will then see **FunctionName** selected. You also see a message about how many **FunctionName** dimension values were found to populate the input box.

10. (Optional) For more settings, choose **Secondary settings** and do one or more of the following:

    - To customize the name of your variable, enter the name in **Custom variable name**.
    - To customize the label for the variable input field, enter the label in **Input label**.
    - To set the default value for this variable when the dashboard is first opened, enter the default in **Default value**.

11. Choose **Add variable**.

    A **FunctionName** dropdown selection box appears near the top of the dashboard. You can select a Lambda function in this box and all the widgets that use the variable will display information about the selected function.

    Later, if you add more widgets to the dashboard that watch Lambda metrics with the **FunctionName** dimension, they will automatically use the variable.

Using widgets on CloudWatch dashboards

You can use the following types of widgets on CloudWatch dashboards:

- **Graph widget** – You can add graphs that contain one or more metrics to your CloudWatch dashboard. The types of graphs that you can add to your dashboard include ***Line***, ***Stacked area***, ***Number***, ***Gauge***, ***Bar***, and ***Pie***. You can remove graphs from your dashboard when you don't need them anymore.
- **Manual metrics widget** – If a metric hasn't published data in the past 14 days, you can't find it when searching for metrics to add to a graph on a CloudWatch dashboard. You can add any metric manually to an existing graph.
- **Metrics explorer widget** – Metrics explorer widgets include graphs of multiple resources that have the same tag, or share the same resource property such as an instance type. These widgets stay up to date, as resources that match are created or deleted. Adding metrics explorer widgets to your dashboard helps you to troubleshoot your environment more efficiently.

  For example, you can monitor your fleet of EC2 instances by assigning tags that represent their environments, such as production or test. You can then use these tags to filter and aggregate the

operational metrics, such as `CPUUtilization`, to understand the health and performance of the EC2 instances that are associated with each tag.

- **Line widget** – With the line widget, you can compare metrics over periods of time. You also can use the widget's mini-map zoom feature to inspect sections of line graphs without changing between zoomed-in and zoomed-out views. The procedures in this section describe how to add and remove a line widget on a CloudWatch dashboard. For information about using the widget's mini-map zoom feature with line graphs, see Zooming in on a line or stacked area graph.
- **Number widget** – With the number widget, you can look at the latest metric values and trends as soon as they appear. Because the number widget includes the sparkline feature, you can visualize the top and bottom halves of metric trends in a single graph. The procedures in this section describe how to add and remove a number widget from a CloudWatch dashboard.
- **Gauge widget** – With the gauge widget, you can visualize metric values that go between ranges. For example, you can use the gauge widget to graph percentages and CPU utilization, so that you can observe and diagnose any performance issues that occur. The procedures in this section describe how to add and remove a gauge widget from a CloudWatch dashboard.
- **Text widget** – A text widget contains a block of text in Markdown format. You can add, edit, or remove text widgets from your CloudWatch dashboard.
- **Alarm widget** – To add an alarm widget to a dashboard, choose one of the following options:
  - Add a single alarm in a widget, which displays the graph of the alarm's metric and also displays the alarm status.
  - Add an *alarm status widget*, which displays the status of multiple alarms in a grid. Only the alarm names and current status are displayed, Graphs are not displayed. You can include up to 100 alarms in one alarm status widget.
- **Table widget** – With the data table widget, you can see the raw datapoints of your metric and a quick summary of that raw data. Because the data table widget is not a chart to abstract the actual data away from you, it is easier to understand the datapoints being presented. The procedures in this section describe how to add and remove a data table widget from a CloudWatch dashboard. The following image shows an example of a table widget with columns for the Minimum, Maximum, Sum, and Average statistics for a set of CloudWatch metrics.
- **Linked graphs** – You can link the graphs on your dashboard together, so that when you zoom in or zoom out on one graph, the other graphs zoom in or zoom out at the same time. You can unlink graphs to limit zoom to one graph.

Topics

- Adding a graph widget to a CloudWatch dashboard
- Removing a graph widget from a CloudWatch dashboard
- Graph metrics manually on a CloudWatch dashboard
- Editing a graph on a CloudWatch dashboard
- Renaming a graph on a CloudWatch dashboard
- Moving a graph on a CloudWatch dashboard
- Changing the size of graphs on CloudWatch dashboards
- Changing the size of a graph temporarily on a CloudWatch dashboard
- Add a metrics explorer widget to a CloudWatch dashboard
- Adding a line graph widget to a CloudWatch dashboard
- Removing a line graph widget from a CloudWatch dashboard
- Adding a number widget to a CloudWatch dashboard
- Removing a number widget from a CloudWatch dashboard
- Adding a gauge widget to a CloudWatch dashboard
- Removing a gauge widget from a CloudWatch dashboard
- Using a custom widget on a CloudWatch dashboard
- Adding a text widget to a CloudWatch dashboard
- Editing a text widget on a CloudWatch dashboard
- Removing a text widget from a CloudWatch dashboard
- Adding an alarm to a CloudWatch dashboard
- Adding an alarm status widget to a CloudWatch dashboard
- Removing an alarm widget from a CloudWatch dashboard
- Using a data table widget in a CloudWatch dashboard
- Linking graphs on a CloudWatch dashboard
- Unlinking graphs on a CloudWatch dashboard

Adding a graph widget to a CloudWatch dashboard

The procedures in this section describe how to add and remove graphs from your dashboard. For information about how to edit a graph on your dashboard, see Edit a graph on a CloudWatch dashboard.

To add a graph to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and then choose the type of graph that you want to add to your dashboard, then choose **Next**.

   - If you select ***Line***, ***Stacked area***, ***Bar***, or ***Pie***, choose **Metrics**.
4. In the **Browse** tab, search or browse for the metrics to graph, and select the ones that you want.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper part of the screen. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. (Optional) To change your graph's widget type, use the dropdown that's next to the predefined time ranges.
7. (Optional) In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

   a. To add a dynamic label, choose **Graphed metrics**, and then choose **Add dynamic labels**. Dynamic labels display a statistic about your metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of your labels. For more information, see Use dynamic labels.

   b. To change the color of a metric, choose the color square that's next to the metric.

   c. To change the statistic, select the dropdown under ***Statistic***, and then choose a new value. For more information, see Statistics.

   d. To change the period, select the dropdown under the ***Period*** column, and then choose a new value.
8. If you are creating a gauge widget, you must choose the **Options** tab and specify the **Min** and **Max** values to use for the two ends of the gauge.

9. (Optional) To customize the Y-axis, choose **Options**. You can add a custom label under ***Left Y-axis*** in the label field. If your graph displays values on the right side of the Y-axis, you can customize that label, too. You also can set minimum and maximum limits on your Y-axis values, so that your graph only displays the value ranges that you specify.
10. (Optional) To add or edit horizontal annotations to line or stacked area graphs, or to add thresholds to gauge widgets, choose **Options**:

    a. To add a horizontal annotation or threshold, choose **Add horizontal annotation** or **Add threshold**.

    b. For ***Label***, enter a label for the annotation then choose the check mark icon.

    c. For ***Value***, choose the pen and paper icon that's next to the current value, and enter your new value. After you enter your value, choose the check mark icon.

    d. For ***Fill***, select the dropdown and specify how your annotation will use shading. You can choose ***None***, ***Above***, ***Between***, or ***Below***. To change the fill color, choose the color square that's next to the annotation.

    e. For ***Axis***, specify whether your annotation appears on the left or right side of the Y-axis.

    f. To hide an annotation, clear the check box that's next to the annotation you want to hide.

    g. To delete an annotation, choose **X** under ***Actions***.

    > **Note**
    >
    > You can repeat these steps to add multiple horizontal annotations or thresholds to the same graph or gauge.

11. (Optional) To add or edit vertical annotations, choose **Options**:

    a. To add a vertical annotation, choose **Add vertical annotation**.

    b. For ***Label***, choose the pen and paper icon that's next to the current annotation, and enter your new annotation. If you want to show only the date and time, leave the label field blank.

    c. For ***Date***, choose the current date and time, and enter the new date and time.

    d. For ***Fill***, select the dropdown, and specify how your annotation will use shading. You can choose ***None***, ***Above***, ***Between***, or ***Below***. To change the fill color, select the color square that's next to the annotation.

    e. To hide an annotation, clear the check box next to the annotation that you want to hide.

f. To delete an annotation, choose **X** under ***Actions***.

> **Note**
>
> You can repeat these steps to add multiple vertical annotations to the same graph.

12. Choose **Create widget**.
13. Choose **Save dashboard**.

Removing a graph widget from a CloudWatch dashboard

To remove a graph from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Graph metrics manually on a CloudWatch dashboard

To add a metric that you can't find in search to a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. The dashboard must already contain a graph where you want to add the metric. If it doesn't, create the graph and add any metric to it. For more information, see Adding a graph widget to a CloudWatch dashboard.
4. Choose **Actions**, **View/edit source**.

   A JSON block appears. The block specifies the widgets on the dashboard and their contents. The following is an example of one part of this block, which defines one graph.

```
{
```

```
  "type": "metric",
  "x": 0,
  "y": 0,
  "width": 6,
  "height": 3,
  "properties": {
    "view": "singleValue",
    "metrics": [
      [
        "AWS/EBS",
        "VolumeReadOps",
        "VolumeId",
        "vol-1234567890abcdef0"
      ]
    ],
    "region": "us-west-1"
  }
},
```

In this example, the following section defines the metric shown on this graph.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ]
```

5. Add a comma after the end bracket if there isn't already one and then add a similar bracketed section after the comma. In this new section, specify the namespace, metric name, and any necessary dimensions of the metric that you're adding to the graph. The following is an example.

```
[ "AWS/EBS", "VolumeReadOps", "VolumeId", "vol-1234567890abcdef0" ],
[ "MyNamespace", "MyMetricName", "DimensionName", "DimensionValue" ]
```

For more information about the formatting of metrics in JSON, see Properties of a Metric Widget Object in the Amazon CloudWatch API Reference.

6. Choose **Update**.

Editing a graph on a CloudWatch dashboard

You can edit the graphs that you add to your CloudWatch dashboard. You can change a graph's title, statistic, or period. You can add, update, and remove metrics from your graphs. If your graph contains more than one metric, you can reduce clutter by hiding metrics that you aren't using. The procedures in this section describe how to edit a graph on your dashboard. For information about creating a graph, see Add or remove a graph from a CloudWatch dashboard.

New interface

To edit a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pen and paper icon that's next to the current title. Enter the new title, and then choose **Apply**.
5. (Optional) To change your graph's time range, select one of the predefined time ranges in the upper are of the graph. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
6. To change your graph's widget type, use dropdown that's next to the predefined time ranges.
7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

   a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see Use dynamic labels.

b. To change the color of a metric, choose the color square that's next to the metric.

c. To change the statistic, choose the statistic value under the ***Statistic*** column, and then choose a new value. For more information, see Statistics.

d. To change the period, choose the period value under the ***Period*** column, and then choose a new value.

8. To add or edit horizontal annotations, choose **Options**:

a. To add a horizontal annotation, choose **Add horizontal annotation**.

b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

c. For **Value**, choose the pen and paper icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.

d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, another new label and value field appears.

> **Tip**
>
> You can change the fill color by choosing the colored square next to the annotation.

e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

g. To delete an annotation, choose **X** under the **Actions** column.

> **Note**
>
> You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Options**:

a. To add a vertical annotation, choose **Add vertical annotation**.

b. For **Label**, choose the pen and paper icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

> **Tip**
>
> To show only the date and time, leave the label field blank.

c. For **Date**, choose the current date and time. Then enter the new date and time.

d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

> **Tip**
>
> You can change the fill color by choosing the color square next to the annotation.

> **Note**
>
> You can repeat these steps to add multiple vertical annotations to the same graph.

e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

f. To delete an annotation, choose **X** under the **Actions** column.

10. To customize the Y-axis, choose **Options**. Under ***Left Y-axis***, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too. You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.

3. In the upper-right corner of the graph that you want to edit, choose **Widget actions**. Choose **Legend** and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An ***X*** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Original interface

To edit a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit. Choose **Widget actions**, and then choose **Edit**.
4. To change the graph's title, choose the pencil icon that's next to the current title, and then enter the new title.
5. To change the graph's time range, choose one of the predefined time ranges in the upper area of the graph. These span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).
   - To set your own time range, choose **custom**.
6. To change your graph's widget type, select the **Graph options** tab. You can choose ***Line***, ***Stacked area***, ***Number***, ***Bar***, or ***Pie***.

> **Tip**
>
> You can change your graph's widget type by choosing the dropdown that's next to the predefined time ranges.

7. In **Graphed metrics**, you can add a dynamic label to your metric and change your metric's label, label color, statistic, and period. You also can determine the position of labels on the Y-axis from left to right.

   a. To add a dynamic label for a metric, choose **Dynamic labels**. Dynamic labels display a statistic about the metric in the graph legend. Dynamic labels update automatically when your dashboard or graph refreshes. By default, the dynamic values that you add to labels appear at the beginning of the labels. For more information, see Use dynamic labels.

   b. To change the color of a metric, choose the color square that's next to the metric.

   c. To change the statistic, choose the statistic value under the ***Statistic*** column, and then choose a new value. For more information, see Statistics.

   d. To change the period, choose the period value under the ***Period*** column, and then choose a new value.

8. To add or edit horizontal annotations, choose **Graph options**:

   a. To add a horizontal annotation, choose **Add horizontal annotation**.

   b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

   c. For **Value**, choose the pencil icon next to the current metric value. Then enter your new metric value. After you enter your value, select the check mark icon.

   d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

   > **Tip**
   >
   > You can change the fill color by choosing the color square next to the annotation.

   e. For **Axis**, specify whether your annotation appears on the left or right side of the Y-axis.

   f. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

   g. To delete an annotation, choose **X** under the **Actions** column.

> **Note**
>
> You can repeat these steps to add multiple horizontal annotations to the same graph.

9. To add or edit vertical annotations, choose **Graph options**:

   a. To add a vertical annotation, choose **Add vertical annotation**.

   b. For **Label**, choose the pencil icon next to the current annotation. Then enter your new annotation. After you enter your annotation, choose the check mark icon.

   > **Tip**
   >
   > To show only the date and time, leave the label field blank.

   c. For **Date**, choose the pencil icon next to the current date and time. Then enter the new date and time.

   d. For **Fill**, choose the dropdown under the column, and then specify how your annotation will use shading. You can choose **None**, **Above**, **Between**, or **Below**. If you choose **Between**, a new label and value field appears.

   > **Tip**
   >
   > You can change the fill color by choosing the color square next to the annotation.

   > **Note**
   >
   > You can repeat these steps to add multiple vertical annotations to the same graph.

   e. To hide an annotation, deselect the check box next to the annotation that you want to hide on the graph.

   f. To delete an annotation, choose **X** under the **Actions** column.

10. To customize the Y-axis, choose **Graph options**. Under ***Left Y-axis***, you can enter a custom label for **Label**. If the graph displays values on the right Y-axis, you can customize that label, too. You also can set minimums and maximums on the Y-axis values, so that the graph displays only the value range that you specify.
11. When you finish making changes, choose **Update widget**.

To hide or change the position of a graph legend

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the graph that you want to edit, and choose **Widget actions**. Choose **Legend**, and select **Hidden**, **Bottom**, or **Right**.

To temporarily hide metrics for a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the color square for the metric that you want to hide in the graph's footer. An ***X*** appears in the color square when you hover over it, and the square turns gray when you choose it.
4. To restore the hidden metric, clear the **X** in the gray square.

Renaming a graph on a CloudWatch dashboard

You can change the default name that CloudWatch assigns to a graph on your dashboard.

To rename a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the title of the graph and choose **Widget actions** and **Edit**.
4. On the **Edit graph** screen, near the top, choose the title of the graph.
5. For **Title**, enter a new name and choose **Ok** (check mark). In the lower-right corner of the **Edit graph** screen, choose **Update widget**.

Moving a graph on a CloudWatch dashboard

To move a graph on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Do one of the following:
   - Hover over the title of the graph until the selection icon appears. Select and drag the graph to a new location on the dashboard.
   - To move the widget to either the top left or bottom left of the dashboard, choose the vertical ellipsis at the upper right of the widget to open the **Widget actions** menu. Then choose **Move**, and choose where to move the widget to.
4. Choose **Save dashboard**.

Changing the size of graphs on CloudWatch dashboards

To change the size of a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To increase or decrease the size, hover over the graph and drag the lower right corner of the graph. Stop dragging the corner when you have the size that you want.
4. Choose **Save dashboard**.

Changing the size of a graph temporarily on a CloudWatch dashboard

To enlarge a graph temporarily

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select the graph. Alternatively, hover over the title of the graph and choose **Widget actions**, **Enlarge**.

Add a metrics explorer widget to a CloudWatch dashboard

The following steps explain how to add a metrics explorer widget to a dashboard using the console. You can also add it programmatically or by using CloudFormation. For more information, see Metrics Explorer Widget Object Definition and AWS::CloudWatch::Dashboard.

To add a metrics explorer widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard where you want to add the metrics explorer widget.
4. Choose the **+** symbol.
5. Choose **Explorer** and then choose **Next**.

   > **Note**
   >
   > You must be opted in to the new dashboard view to be able to add a Metrics Explorer widget. To opt in, choose **Dashboards** in the navigation pane, then choose **try out the new interface** in the banner at the top of the page.

6. Do one of the following:
   - To use a template, choose **Pre-filled Explorer widget** and then select a template to use.
   - To create a custom visualization, choose **Empty Explorer widget**.
7. Choose **Create**.

   If you used a template, the widget appears on your dashboard with the selected metrics. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

   If you did not use a template, continue to the following steps.
8. In the new widget under **Explorer**, in the **Metrics** box, choose a single metric or all the available metrics from a service.

   After you choose a metric, you can optionally repeat this step to add more metrics.
9. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name and then choose the statistic that you want.
10. Under **From**, choose a tag or a resource property to filter your results.

After you do this, you can optionally repeat this step to choose more tags or resource properties.

If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

If you choose different properties or tags, such as the **`Production`** tag and the M5 instance type, only the resources that match all of these selections are displayed. This is treated as an AND operation.

11. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

    Depending on how you choose to aggregate, the result may be a single time series or multiple time series.
12. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
13. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.
14. If you're satisfied with the explorer widget and the dashboard, choose **Save dashboard**.

Adding a line graph widget to a CloudWatch dashboard

To add a line widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Line**.
4. Choose **Metrics**.
5. Choose **Browse**, and select the metric that you want to graph.
6. Choose **Create widget**, and then choose **Save dashboard**.

Removing a line graph widget from a CloudWatch dashboard

To remove a line graph widget from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the line widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Adding a number widget to a CloudWatch dashboard

> **Note**
>
> Only the new interface supports the sparkline feature. When you create a number widget, the sparkline feature is automatically included.

To add a number widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol, and select **Number**.
4. In the **Browse** tab, search or browse for the metric that you want to display.
5. (Optional) To change the color of the sparkline feature, choose **Graphed metrics**, and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
6. (Optional) To turn off the sparkline feature, choose **Options**. Under ***Sparkline***, the check box.
7. (Optional) To change your number widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget keep using this time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.

8. (Optional) To have the number widget display an aggregate (***1h***, ***3h***, ***12h***, ***1d***, ***3d***, or ***1w***).

   To set your own time range, choose **Custom**.

   - (Optional) To have this widget display an average of the metric value over the entire time range, instead of the most recent value, choose **Options**, **Time range value shows the value from the entire time range**.

9. Choose **Create widget**, and choose **Save dashboard**.

> **Tip**
>
> You can turn off the sparkline feature from the number widget on the dashboard screen. In the upper-right corner of the number widget that you want to modify, choose **Widget actions**. Select **Sparkline**, and then choose **Hide sparkline**.

Removing a number widget from a CloudWatch dashboard

To remove a number widget from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the number widget that you want to delete.
3. In the upper-right corner of the number widget that you want to remove, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**.

Adding a gauge widget to a CloudWatch dashboard

> **Note**
>
> Only the new interface in the CloudWatch console supports creation of the gauge widget. You must set a gauge range when you create this widget.

To add a gauge widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, choose the **+** symbol, and then select **Gauge**.
4. Choose **Browse**, and then select the metric that you want to graph.
5. Choose **Options**. Under ***Gauge range***, set values for **Min** and **Max**. For percentages, such as CPU utilization, we recommend that you set the values for `Min` to `0` and `Max` to `100`.
6. (Optional) To change the color of the gauge widget, choose **Graphed metrics** and select the color box next to the metric label. A menu appears where you can choose a different color or enter a six-digit hex color code to specify a color.
7. Choose **Create widget**, and choose **Save dashboard**.

Removing a gauge widget from a CloudWatch dashboard

To remove a gauge widget from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose the dashboard that contains the gauge widget you want to delete.
3. In the upper-right corner of the gauge widget that you want to delete, choose **Widget actions**, and choose **Delete**.
4. Choose **Save dashboard**.

Using a custom widget on a CloudWatch dashboard

A *custom widget* is a CloudWatch dashboard widget that can call any AWS Lambda function with custom parameters. It then displays the returned HTML or JSON. Custom widgets are a simple way to build a custom data view on a dashboard. If you can write Lambda code and create HTML, you can create a useful custom widget. Additionally, Amazon provides several prebuilt custom widgets that you can create without any code.

When you create a Lambda function to use as a custom widget, we strongly recommend that you include the prefix **customWidget** in the function name. This helps you know which of your Lambda functions are safe to use when you add custom widgets to your dashboard.

Custom widgets behave like other widgets on your dashboard. They can be refreshed and auto-refreshed, resized, and moved around. They react to the time range of the dashboard.

If you have set up CloudWatch console cross-account functionality, you can add a custom widget created in one account to dashboards in other accounts. For more information, see Cross-account cross-Region CloudWatch console.

You can also use custom widgets on your own website by using the CloudWatch dashboard sharing feature. For more information, see Sharing CloudWatch dashboards.

Topics

- Details about custom widgets in CloudWatch
- Security and JavaScript for custom CloudWatch widgets
- Interactivity in the custom widget in CloudWatch
- Creating a custom widget for a CloudWatch dashboard
- Sample custom widgets for a CloudWatch dashboard

Details about custom widgets in CloudWatch

Custom widgets work as follows:

1. The CloudWatch dashboard calls the Lambda function containing the widget code. It passes in any custom parameters that are defined in the widget.
2. The Lambda function returns a string of HTML, JSON, or Markdown. Markdown is returned as JSON in the following format:

   ```
   {"markdown":"markdown content"}
   ```

3. The dashboard displays the returned HTML or JSON.

If the function returns HTML, most HTML tags are supported. You can use Cascading Style Sheets (CSS) styles and Scalable Vector Graphics (SVG) to build sophisticated views.

The default style of HTML elements such as links and tables follow the styling of CloudWatch dashboards. You can customize this style by using inline styles, using the `<style>` tag. You can also deactivate the default styles by including a single HTML element with the class of `cwdb-no-`

`default-styles`. The following example deactivates default styles: `<div class="cwdb-no-default-styles"></div>`.

Every call by a custom widget to Lambda includes a `widgetContext` element with the following contents, to provide the Lambda function developer with useful context information.

```
{
  "widgetContext": {
    "dashboardName": "Name-of-current-dashboard",
    "widgetId": "widget-16",
    "accountId": "012345678901",
    "locale": "en",
    "timezone": {
      "label": "UTC",
      "offsetISO": "+00:00",
      "offsetInMinutes": 0
    },
    "period": 300,
    "isAutoPeriod": true,
    "timeRange": {
      "mode": "relative",
      "start": 1627236199729,
      "end": 1627322599729,
      "relativeStart": 86400012,
      "zoom": {
        "start": 1627276030434,
        "end": 1627282956521
      }
    },
    "theme": "light",
    "linkCharts": true,
    "title": "Tweets for Amazon website problem",
    "forms": {
      "all": {}
    },
    "params": {
      "original": "param-to-widget"
    },
    "width": 588,
    "height": 369
  }
}
```

Default CSS styling

Custom widgets provide the following default CSS styling elements:

- You can use the CSS class **btn** to add a button. It turns an anchor (<a>) into a button as in the following example:

```
<a class="btn" href=https://amazon.com”>Open Amazon</a>
```

- You can use the CSS class **btn btn-primary** to add a primary button.
- The following elements are styled by default: **table**, **select**, **headers (h1, h2, and h3)**, **preformatted text (pre)**, **input**, and **text area**.

Using the describe parameter

We strongly recommend that you support the **describe** parameter in your functions, even if it just returns an empty string. If you don't support it, and it is called in the custom widget, it displays widget content as if it was documentation.

If you include the **describe** parameter, the Lambda function returns the documentation in Markdown format and does nothing else.

When you create a custom widget in the console, after you select the Lambda function a **Get documentation** button appears. If you choose this button, the function is invoked with the **describe** parameter and the function's documentation is returned. If the documentation is well-formatted in markdown, CloudWatch parses the first entry in the documentation that is surrounded by three single backtick characters (```) in YAML. Then, it automatically populates the documentation in the parameters. The following is an example of such well-formatted documentation.

```
``` yaml
echo: <h1>Hello world</h1>
```
```

Security and JavaScript for custom CloudWatch widgets

For security reasons, JavaScript is not allowed in the returned HTML. Removing the JavaScript prevents permission escalation issues, where the writer of the Lambda function injects code that could run with higher permissions than the user viewing the widget on the dashboard.

If the returned HTML contains any JavaScript code or other known security vulnerabilities, it is cleaned from the HTML before it is rendered on the dashboard. For example, the **<iframe>** and **<use>** tags are not allowed and are removed.

Custom Widgets won't run by default in a dashboard. Instead, you must explicitly allow a custom widget to run if you trust the Lambda function that it invokes. You can choose to allow it once or allow always, for both individual widgets and entire dashboard. You can also deny permission for individual widgets and the entire dashboard.

Interactivity in the custom widget in CloudWatch

Even though JavaScript is not allowed, there are other ways to allow interactivity with the returned HTML.

- Any element in the returned HTML can be tagged with special configuration in a `<cwdb-action>` tag, which can display information in pop-ups, ask for confirmation on clicks, and call any Lambda function when that element is chosen. For example, you can define buttons that call any AWS API using a Lambda function. The returned HTML can be set to either replace the existing Lambda widget's content, or display inside a modal.
- The returned HTML can include links that open new consoles, open other customer pages, or load other dashboards.
- The HTML can include the `title` attribute for an element, which gives additional information if the user hovers over that element.
- The element can include CSS selectors, such as `:hover`, which can invoke animations or other CSS effects. You can also show or hide elements in the page.

<cwdb-action> definition and usage

The `<cwdb-action>` element defines a behavior on the immediately previous element. The content of the `<cwdb-action>` is either HTML to display or a JSON block of parameters to pass to a Lambda function.

The following is an example of a `<cwdb-action>` element.

```
<cwdb-action
     action="call|html"
     confirmation="message"
     display="popup|widget"
     endpoint="<lambda ARN>"
     event="click|dblclick|mouseenter">

     html | params in JSON
</cwdb-action>
```

- **action**— Valid values are `call`, which calls a Lambda function, and `html`, which displays any HTML contained within `<cwdb-action>`. The default is `html`.
- **confirmation**— Displays a confirmation message that must be acknowledged before the action is taken, allowing the customer to cancel.
- **display**— Valid values are popup and `widget`, which replaces the content of the widget itself. The default is `widget`.
- **endpoint**— The Amazon Resource Name (ARN) of the Lambda function to call. This is required if `action` is `call`.
- **event**— Defines the event on the previous element that invokes the action. Valid values are `click`, `dblclick`, and `mouseenter`. The `mouseenter` event can be used only in combination with the `html` action. The default is `click`.

Examples

The following is an example of how to use the `<cwdb-action>` tag to create a button that reboots an Amazon EC2 instance using a Lambda function call. It displays the success or failure of the call in a pop-up.

```
<a class="btn">Reboot Instance</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:rebootInstance" display="popup">
       { "instanceId": "i-342389adbfef" }
</cwdb-action>
```

The next example displays more information in a pop-up.

```
<a>Click me for more info in popup</a>
<cwdb-action display="popup">
```

```
   <h1>Big title</h1>
   More info about <b>something important</b>.
</cwdb-action>
```

This example is a **Next** button that replaces the content of a widget with a call to a Lambda function.

```
<a class="btn btn-primary">Next</a>
<cwdb-action action="call" endpoint="arn:aws:lambda:us-
east-1:123456:function:nextPage">
   { "pageNum": 2 }
</cwdb-action>
```

Creating a custom widget for a CloudWatch dashboard

To create a custom widget, you can use one of the samples provided by AWS, or you can create your own. The AWS samples include samples in both JavaScript and Python, and are created by a AWS CloudFormation stack. For a list of samples, see Sample custom widgets for a CloudWatch dashboard.

To create a custom widget in a CloudWatch dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol.
4. Choose **Custom widget**.
5. Use one of the following methods:
   - To use a sample custom widget provided by AWS, do the following:
     a. Select the sample in the dropdown box.

        The CloudFormation console launches in a new browser. In the CloudFormation console, do the following:
     b. (Optional) Customize the CloudFormation stack name.
     c. Make selections for any parameters used by the sample.
     d. Select **I acknowledge that AWS CloudFormation might create IAM resources**, and choose **Create stack**.
   - To create your own custom widget provided by AWS, do the following:

a. Choose **Next**.

b. Choose to either select your Lambda function from a list, or enter its Amazon Resource Name (ARN). If you select it from a list, also specify the Region where the function is and the version to use.

c. For **Parameters**, make selections for any parameters used by the function.

d. Enter a title for the widget.

e. For **Update on**, configure when the widget should be updated (when the Lambda function should be called again). This can be one or more of the following: **Refresh** to update it when the dashboard auto-refreshes, **Resize** to update it whenever the widget is resized, or **Time Range** to update it whenever the dashboard's time range is adjusted, including when graphs are zoomed into.

f. If you are satisfied with the preview, choose **Create widget**.

Sample custom widgets for a CloudWatch dashboard

AWS provides sample custom widgets in both JavaScript and Python. You can create these sample widgets by using the link for each widget in this list. Alternatively, you can create and customize a widget by using the CloudWatch console. The links in this list open an AWS CloudFormation console and use an CloudFormation quick-create link to create the custom widget.

You can also access the custom widget samples on GitHub.

Following this list, complete examples of the Echo widget are shown for each language.

JavaScript

Sample custom widgets in JavaScript

- Echo – A basic echoer that you can use to test how HTML appears in a custom widget, without having to write a new widget.
- Hello world – A very basic starter widget.
- Custom widget debugger – A debugger widget that displays useful information about the Lambda runtime environment.
- Query CloudWatch Logs Insights – Run and edit CloudWatch Logs Insights queries.
- Run Amazon Athena queries – Run and edit Athena queries.
- Call AWS API – Call any read-only AWS API and display the results in JSON format.

- Fast CloudWatch bitmap graph – Render CloudWatch graphs using on the server side, for fast display.
- Text widget from CloudWatch dashboard – Displays the first text widget from the specified CloudWatch dashboard.
- CloudWatch metric data as a table – Displays raw CloudWatch metric data in a table.
- Amazon EC2 table – Displays the top EC2 instances by CPU utilization. This widget also includes a Reboot button, which is disabled by default.
- AWS CodeDeploy deployments – Displays CodeDeploy deployments.
- AWS Cost Explorer report – Displays a report on the cost of each AWS service for a selected time range.
- Display content of external URL – Displays the content of an externally accessible URL.
- Display an Amazon S3 object – Displays an object in an Amazon S3 bucket in your account.
- Simple SVG pie chart – Example of a graphical SVG-based widget.

Python

Sample custom widgets in Python

- Echo – A basic echoer which you can use to test how HTML appears in a custom widget, without having to write a new widget.
- Hello world – A very basic starter widget.
- Custom widget debugger – A debugger widget that displays useful information about the Lambda runtime environment.
- Call AWS API – Call any read-only AWS API and display the results in JSON format.
- Fast CloudWatch bitmap graph – Render CloudWatch graphs using on the server side, for fast display.
- Send dashboard snapshot by email – Take a snapshot of the current dashboard and send it to email recipients.
- Send dashboard snapshot to Amazon S3 – Take a snapshot of the current dashboard and store it in Amazon S3.
- Text widget from CloudWatch dashboard – Displays the first text widget from the specified CloudWatch dashboard.
- Display content of external URL – Displays the content of an externally accessible URL.

- RSS reader – Displays RSS feeds.
- Display an Amazon S3 object – Displays an object in an Amazon S3 bucket in your account.
- Simple SVG pie chart – Example of a graphical SVG-based widget.

Echo widget in JavaScript

The following is the Echo sample widget in JavaScript.

```
const DOCS = `
## Echo
A basic echo script. Anything passed in the \`\`\`echo\`\`\` parameter is returned as
 the content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
\`\`\` yaml
echo: <h1>Hello world</h1>
\`\`\`
`;

exports.handler = async (event) => {
    if (event.describe) {
        return DOCS;
    }

    let widgetContext = JSON.stringify(event.widgetContext, null, 4);
    widgetContext = widgetContext.replace(/</g, '&lt;');
    widgetContext = widgetContext.replace(/>/g, '&gt;');

    return `${event.echo || ''}<pre>${widgetContext}</pre>`;
};
```

Echo widget in Python

The following is the Echo sample widget in Python.

```
import json

DOCS = """
```

```
## Echo
A basic echo script. Anything passed in the ```echo``` parameter is returned as the
 content of the custom widget.
### Widget parameters
Param | Description
---|---
**echo** | The content to echo back

### Example parameters
``` yaml
echo: <h1>Hello world</h1>
```"""

def lambda_handler(event, context):
    if 'describe' in event:
        return DOCS

    echo = event.get('echo', '')
    widgetContext = event.get('widgetContext')
    widgetContext = json.dumps(widgetContext, indent=4)
    widgetContext = widgetContext.replace('<', '<')
    widgetContext = widgetContext.replace('>', '>')

    return f'{echo}<pre>{widgetContext}</pre>'
```

Echo widget in Java

The following is the Echo sample widget in Java.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class Handler implements RequestHandler<Event, String>{

  static String DOCS = ""
    + "## Echo\n"
    + "A basic echo script. Anything passed in the ```echo``` parameter is returned as
 the content of the custom widget.\n"
    + "### Widget parameters\n"
```

```
    + "Param | Description\n"
    + "---|---\n"
    + "**echo** | The content to echo back\n\n"
    + "### Example parameters\n"
    + "```yaml\n"
    + "echo: <h1>Hello world</h1>\n"
    + "```\n";

  Gson gson = new GsonBuilder().setPrettyPrinting().create();

  @Override
  public String handleRequest(Event event, Context context) {

    if (event.describe) {
      return DOCS;
    }

    return (event.echo != null ? event.echo : "") + "<pre>" +
 gson.toJson(event.widgetContext) + "</pre>";
  }
}

class Event {

    public boolean describe;
    public String echo;
    public Object widgetContext;

    public Event() {}

    public Event(String echo, boolean describe, Object widgetContext) {
        this.describe = describe;
        this.echo = echo;
        this.widgetContext = widgetContext;
    }
}
```

Adding a text widget to a CloudWatch dashboard

To add a text widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.

3. Choose the **+** symbol.
4. Choose **Text**.
5. For **Markdown**, add and format your text using Markdown and choose **Create widget**.
6. To make the text widget transparent, choose **Transparent background**.
7. Choose **Save dashboard**.

Editing a text widget on a CloudWatch dashboard

To edit a text widget on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Edit**.
4. Update the text as needed and choose **Update widget**.
5. Choose **Save dashboard**.

Removing a text widget from a CloudWatch dashboard

To remove a text widget from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the upper-right corner of the text block and choose **Widget actions**. Then, choose **Delete**.
4. Choose **Save dashboard**.

Adding an alarm to a CloudWatch dashboard

To add a single alarm, including its graph, to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, select the alarm to add, and then choose **Add to Dashboard**.

3. Select a dashboard, choose a widget type (**Line**, **Stacked area**, or **Number**), and then choose **Add to dashboard**.
4. To see your alarm on the dashboard, choose **Dashboards** in the navigation pane and select the dashboard.
5. (Optional) To temporarily make an alarm graph larger, select the graph.
6. (Optional) To change the widget type, hover over the title of the graph, choose **Widget actions**, and then choose **Widget type**.

Adding an alarm status widget to a CloudWatch dashboard

To add an alarm status widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose the **+** symbol.
4. Choose **Alarm status**.
5. Select the check boxes next to the alarms that you want to add to the widget, and then choose **Create widget**.
6. Choose **Add to dashboard**.

Removing an alarm widget from a CloudWatch dashboard

To remove an alarm widget from a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Hover over the widget, choose **Widget actions**, and then choose **Delete**.
4. Choose **Save dashboard**. If you attempt to navigate away from the dashboard before you save your changes, you're prompted to either save or discard your changes.

Using a data table widget in a CloudWatch dashboard

Table properties

A data table has a default set of properties that don't require any changes to be made to the options or source. These properties include a sticky label column, all summary columns enabled, datapoints rounded, and their units converted.

Each data table widget can have the following properties. The information about each property includes how to configure it in the JSON source of the dashboard. For more information about dashboard JSON, see Dashboard Body Structure and Syntax.

Summary

Summary columns are a new property introduced with the data table widget. These columns are a specific subset of summaries of your current table. For example, the **Sum** summary is a sum of all displayed datapoints in its row. The summary columns are not the same as CloudWatch statistics. Represented in source as:

```
 "table": {
        "summaryColumns": [
            "MIN",
            "MAX",
            "SUM",
            "AVG"
        ]
    },
```

Thresholds

Use this to apply thresholds to your table. When a data point falls within a threshold, its cell is highlighted with the threshold color. Represented in source as:

```
"annotations": {
    "horizontal": [
        {
            "label": string,
            "value": int,
            "fill": "above" | "below"
        }
    ]
}
```

Unit in label column

To display what unit is associated with the metric, you can enable this option to display the unit in the label column beside the label. Represented in source as:

```
"yAxis": {
    "left": {
        "showUnits": true | false
    }
}
```

Invert rows and columns

This transforms the table so that the datapoints swap from columns to rows, and the metrics become columns. Represented in source as:

```
 "table": {
    "layout": "vertical" | "horizontal"
}
```

Sticky summary columns

This makes the summary columns sticky, so that they remain in view while you scroll. The label is already sticky. Represented in source as:

```
"table": {
    "stickySummary": true | false
}
```

Display only summary columns

This prevents the columns of datapoints from being displayed, so that only the label and summary columns are displayed. Represented in source as:

```
 "table": {
    "showTimeSeriesData": false | true
}
```

Live data

Displays the most recent data point, even if it is not yet fully aggregated. Represented in source as:

```
"liveData": true | false
```

Number widget format

Displays as many digits as can fit in the cell, before rounding and converting. Represented in source as:

```
"singleValueFullPrecision": true | false
```

Adding a data table widget to a CloudWatch dashboard

To add a data table widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards** and then choose a dashboard.
3. Choose the **+** button, select **Data table**, and choose **Next**.
4. In the **Browse** tab, search or browse for the metrics that you want to display in the table widget. Then select the metrics.
5. (Optional) To change the layout of the table, choose the **Options** tab and select **Invert rows and columns**.

   You can also use the **Options** tab to change what columns appear in the table and display the unit being used in the **Label** column.

   > **Tip**
   >
   > To display more accurate thresholds, choose **Show as many digits as can fit before rounding**.

6. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.
7. (Optional) To change your data table widget's time range, select one of the predefined time ranges in the upper area of the widget. The time ranges span from 1 hour to 1 week. To set your own time range, choose **Custom**.

8. (Optional) To have this widget keep using the time range that you select, even if the time range for the rest of the dashboard is later changed, choose **Persist time range**.
9. Choose **Create widget** and then choose **Save dashboard**.

Removing a data table widget from a CloudWatch dashboard

To remove a data table widget to a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the widget that you want to remove, choose **Widget actions**, **Delete**.
4. Choose **Save dashboard**.

Linking graphs on a CloudWatch dashboard

Link graphs on a CloudWatch dashboard so that changes in time are reflected on all graphs.

To link the graphs on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Choose **Actions** and then **Link graphs**.

Unlinking graphs on a CloudWatch dashboard

To unlink the graphs on a dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Clear **Actions** and then **Link graphs**.

Sharing CloudWatch dashboards

You can share your CloudWatch dashboards with people who do not have direct access to your AWS account. This enables you to share dashboards across teams, with stakeholders, and with people external to your organization. You can even display dashboards on big screens in team areas, or embed them in Wikis and other webpages.

Warning

All people who you share the dashboard with are granted the permissions listed in Permissions that are granted to people who you share the dashboard with for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.
The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share dashboards, you can designate who can view the dashboard in three ways:

- Share a single dashboard and designate as many as five email addresses of people who can view the dashboard. Each of these users creates their own password that they must enter to view the dashboard.
- Share a single dashboard publicly, so that anyone who has the link can view the dashboard.
- Share all the CloudWatch dashboards in your account and specify a third-party single sign-on (SSO) provider for dashboard access. All users who are members of this SSO provider's list can access all the dashboards in the account. To enable this, you integrate the SSO provider with Amazon Cognito. The SSO provider must support Security Assertion Markup Language (SAML). For more information about Amazon Cognito, see What is Amazon Cognito?

Sharing a dashboard doesn't incur charges, but widgets inside a shared dashboard incur charges at standard CloudWatch rates. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

When you share a dashboard, Amazon Cognito resources are created in the US East (N. Virginia) Region.

Important

Do not modify resource names and identifiers that are created by the dashboard sharing process. This includes Amazon Cognito and IAM resources. Modifying these resources can cause unexpected and incorrect functionality of shared dashboards.

Note

If you share a dashboard that has metric widgets with alarm annotations, the people that you share the dashboard with will not see those widgets. They will instead see a blank widget with text saying that the widget is not available. You will still see metric widgets with alarm annotations when you view the dashboard yourself.

Permissions required to share a dashboard

To be able to share dashboards using any of the following methods and to see which dashboards have already been shared, you must be signed on as a user or with an IAM role that has certain permissions.

To share dashboards, your IAM user or role must have the permissions in the following policy statement:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CWDBSharing*",
                "arn:aws:iam::*:policy/*"
            ]
        },
```

```
        {
            "Sid": "RequiredCognitoPermissionsForDashboardSharing",
            "Effect": "Allow",
            "Action": [
                "cognito-idp:*",
                "cognito-identity:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetDashboard"
            ],
            "Resource": [
                "arn:aws:cloudwatch::account-id:dashboard/dashboard-name"
            ]
        }
    ]
}
```

In the last statement, you can specify * instead of a dashboard ARN to grant access to all dashboards in the account.

> **Note**
>
> The CloudWatch dashboard sharing feature requires the `cognito-idp:*` and `cognito-identity:*` permissions with `Resource` set to `*`. You cannot scope these permissions to specific actions or resources because the dashboard sharing process automatically creates Amazon Cognito resources, and creation actions cannot target resources that do not yet exist.

To be able to see which dashboards are shared, but not be able to share dashboards, a user or an IAM role can include a policy statement similar to the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "RequiredCognitoPermissionsForDashboardSharing",
            "Effect": "Allow",
            "Action": [
                "cognito-idp:*",
                "cognito-identity:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListDashboards"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Note

The CloudWatch dashboard sharing feature requires the `cognito-idp:*` and `cognito-identity:*` permissions with `Resource` set to `*`. You cannot scope these permissions to specific actions or resources because the dashboard sharing process automatically creates Amazon Cognito resources, and creation actions cannot target resources that do not yet exist. The `cloudwatch:ListDashboards` action does not support resource-level permissions, so its `Resource` must also be `*`.

Permissions that are granted to people who you share the dashboard with

When you share a dashboard, CloudWatch creates an IAM role in the account which gives the following permissions to the people who you share the dashboard with:

- `cloudwatch:GetInsightRuleReport`

- `cloudwatch:GetMetricData`
- `cloudwatch:DescribeAlarms`
- `ec2:DescribeTags`

Warning

All people who you share the dashboard with are granted these permissions for the account. If you share the dashboard publicly, then everyone who has the link to the dashboard has these permissions.
The `cloudwatch:GetMetricData` and `ec2:DescribeTags` permissions cannot be scoped down to specific metrics or EC2 instances, so the people with access to the dashboard can query all CloudWatch metrics and the names and tags of all EC2 instances in the account.

When you share a dashboard, by default the permissions that CloudWatch creates restrict access to only the alarms and Contributor Insights rules that are on the dashboard when it is shared. If you add new alarms or Contributor Insights rules to the dashboard and want them to also be seen by the people who you shared the dashboard with, you must update the policy to allow these resources.

Sharing a CloudWatch dashboard with specific users

Use the steps in this section to share a dashboard with as many as five email addresses that you choose.

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.
By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share a dashboard with specific users

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Next to **Share your dashboard and require a username and password**, choose **Start sharing**.
6. Under **Add email addresses**, enter the email addresses that you want to share the dashboard with. You can include as many as five email addresses.
7. When you have all the email addresses entered, read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then paste this link into email and send it to the invited users. They automatically receive a separate email with their user name and a temporary password to use to connect to the dashboard.

Sharing a CloudWatch dashboard publicly

Follow the steps in this section to share a dashboard publicly. This can be useful to display the dashboard on a big screen in a team room, or embed it in a Wiki page.

> **Important**
>
> Sharing a dashboard publicly makes it accessible to anyone who has the link, with no authentication. Do this only for dashboards that do not contain sensitive information.

> **Note**
>
> By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.

By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share a dashboard publicly

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Next to **Share your dashboard publicly**, choose **Start sharing**.
6. Enter **`Confirm`** in the text box.
7. Read the agreement and select the confirmation box. Then choose **Preview policy**.
8. Confirm that the resources that will be shared are what you want, and choose **Confirm and generate shareable link**.
9. On the next page, choose **Copy link to clipboard**. You can then share this link. Anyone you share the link with can access the dashboard, without providing credentials.

Sharing all CloudWatch dashboards in the account by using SSO

Use the steps in this section to share all the dashboards in your account with users by using single sign-on (SSO).

Note

By default, any CloudWatch Logs widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see logs table widgets.
By default, any composite alarm widgets on the dashboard are not visible to people who you share the dashboard with. For more information, see Allowing people that you share with to see composite alarms.

To share your CloudWatch dashboards with users who are in an SSO provider's list

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of your dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Choose **Go to CloudWatch Settings**.
6. If the SSO provider that you want isn't listed in **Available SSO providers**, choose **Manage SSO providers** and follow the instructions in Setting up SSO for CloudWatch dashboard sharing.

   Then return to the CloudWatch console and refresh the browser. The SSO provider that you enabled should now appear in the list.
7. Choose the SSO provider that you want in the **Available SSO providers** list.
8. Choose **Save changes**.

Setting up SSO for CloudWatch dashboard sharing

To set up dashboard sharing through a third-party single sign-on provider that supports SAML, follow these steps.

> **Important**
>
> We strongly recommend that you do not share dashboards using a non-SAML SSO provider. Doing so causes a risk of inadvertently allowing third parties to access your account's dashboards.

To set up an SSO provider to enable dashboard sharing

1. Integrate the SSO provider with Amazon Cognito. For more information, see Integrating Third-Party SAML Identity Providers with Amazon Cognito User Pools.
2. Download the metadata XML file from your SSO provider.
3. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
4. In the navigation pane, choose **Settings**.
5. In the **Dashboard sharing** section, choose **Configure**.

6. Choose **Manage SSO providers**.

   This opens the Amazon Cognito console in the US East (N. Virginia) Region (us-east-1). If you don't see any **User Pools**, the Amazon Cognito console might have opened in a different Region. If so, change the Region to **US East (N. Virginia) us-east-1** and proceed with the next steps.
7. Choose the **CloudWatchDashboardSharing** pool.
8. In the navigation pane, choose **Social and external providers**.
9. Choose **Add identity provider**.
10. Choose **SAML**.
11. Enter a name for your SSO provider in **Provider name**.
12. Choose **Select file**, and select the metadata XML file that you downloaded in step 2.
13. Choose **Create provider**.

Seeing how many of your CloudWatch dashboards are shared

You can use the CloudWatch console to see how many of your CloudWatch dashboards are currently being shared with others.

To see how many of your dashboards are being shared

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. The **Dashboard sharing** section displays how many dashboards are shared.
4. To see which dashboards are shared, choose ***number*** **dashboards shared** under **Username and password** and under **Public dashboards.**

Seeing which of your CloudWatch dashboards are shared

You can use the CloudWatch console to see which of your dashboards are currently being shared with others.

To see which of your dashboards are being shared

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Dashboards**.
3. In the list of dashboards, see the **Share** column. Dashboards that have the icon in this column filled in are currently being shared.
4. To see which users a dashboard is being shared with, choose the dashboard name, and then choose **Actions**, **Share dashboard**.

   The **Share dashboard** ***`dashboard name`*** page displays how the dashboard is being shared. If you want, you can stop sharing the dashboard by choosing **Stop sharing**.

Stopping sharing one or more CloudWatch dashboards

You can stop sharing a single shared dashboard, or stop sharing all shared dashboards at once.

To stop sharing a single dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Choose **Stop sharing**.
6. In the confirmation box, choose **Stop sharing**.

To stop sharing all shared dashboards

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. In the **Dashboard sharing** section, choose **Stop sharing all dashboards**.
4. In the confirmation box, choose **Stop sharing all dashboards**.

Reviewing shared CloudWatch dashboard permissions and changing permission scope

Use the steps in this section if you want to review the permissions of the users of your shared dashboards, or change the scope of shared dashboard permissions.

To review shared dashboard permissions

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. (Optional) To limit which alarms that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:DescribeAlarms` permission from its current position to a new `Allow` statement that lists the ARNs of only the alarms that you want to be seen by shared dashboard users. See the following example.

   ```
   {
      "Effect": "Allow",
       "Action": "cloudwatch:DescribeAlarms",
       "Resource": [
           "AlarmARN1",
           "AlarmARN2"
       ]
   }
   ```

   If you do this, be sure to remove the `cloudwatch:DescribeAlarms` permission from a section of the current policy that looks like this:

   ```
   {
      "Effect": "Allow",
       "Action": [
           "cloudwatch:GetInsightRuleReport",
           "cloudwatch:GetMetricData",
           "cloudwatch:DescribeAlarms",
           "ec2:DescribeTags"
       ],
       "Resource": "*"
   }
   ```

8. (Optional) To limit the scope of what Contributor Insights rules that shared dashboard users can see, choose **Edit policy** and move the `cloudwatch:GetInsightRuleReport` from its

current position to a new `Allow` statement that lists the ARNs of only the Contributor Insights rules that you want to be seen by shared dashboard users. See the following example.

```
{
   "Effect": "Allow",
    "Action": "cloudwatch:GetInsightRuleReport",
    "Resource": [
        "PublicContributorInsightsRuleARN1",
        "PublicContributorInsightsRuleARN2"
    ]
}
```

If you do this, be sure to remove `cloudwatch:GetInsightRuleReport` from a section of the current policy that looks like this:

```
{
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport",
                "cloudwatch:GetMetricData",
                "cloudwatch:DescribeAlarms",
                "ec2:DescribeTags"
            ],
            "Resource": "*"
        }
```

Allowing people that you share with to see composite alarms

When you share a dashboard, by default the composite alarm widgets on the dashboard are not visible to the people who you share the dashboard with. For composite alarm widgets to be visible, you need to add a `DescribeAlarms: *` permission to the dashboard sharing policy. That permission would look like this:

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
}
```

Warning

The preceding policy statement gives access to all alarms in the account. To reduce the scope of `cloudwatch:DescribeAlarms`, you must use a `Deny` statement. You can add a `Deny` statement to the policy and specify the ARNs of the alarms that you want to lock down. That deny statement should look similar to the following:

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": "*"
},
{
    "Effect": "Deny",
    "Action": "cloudwatch:DescribeAlarms",
    "Resource": [
        "SensitiveAlarm1ARN",
        "SensitiveAlarm1ARN"
    ]
}
```

Allow widgets that use PromQL queries to be visible in shared dashboards

When you share a dashboard, widgets that use PromQL (Prometheus Query Language) queries are not visible by default to people with whom you share the dashboard. If you want these people to see PromQL widgets, you must add permissions to the IAM role for dashboard sharing.

To allow people that you share a dashboard with to see PromQL widgets

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement.

```
{
    "Effect": "Allow",
    "Action": "cloudwatch:ListMetrics",
    "Resource": "*"
}
```

8. Choose **Save Changes**.

Allowing people that you share with to see logs table widgets

When you share a dashboard, by default the CloudWatch Logs Insights widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both CloudWatch Logs Insights widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see CloudWatch Logs widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the CloudWatch Logs widgets

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.
4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the log groups that you want shared. See the following example.

```
{
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetLogRecord",
                "logs:GetQueryResults",
```

```
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "SharedLogGroup1ARN",
                "SharedLogGroup2ARN"
            ]
        },
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes the preceding permissions with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the log groups that you want shared. For example, if your `Resource` section for these permissions was the following:

```
"Resource": "*"
```

Change the policy to specify only the ARNs of the log groups that you want shared, as in the following example:

```
"Resource": [
  "SharedLogGroup1ARN",
  "SharedLogGroup2ARN"
]
```

Allowing people that you share with to see custom widgets

When you share a dashboard, by default the custom widgets that are on the dashboard are not visible to the people who you share the dashboard with. This affects both custom widgets that exist now and any that are added to the dashboard after you share it.

If you want these people to be able to see custom widgets, you must add permissions to the IAM role for dashboard sharing.

To allow the people that you share a dashboard with to see the custom widgets

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the shared dashboard.

4. Choose **Actions**, **Share dashboard**.
5. Under **Resources**, choose **IAM Role**.
6. In the IAM console, choose the displayed policy.
7. Choose **Edit policy** and add the following statement. In the new statement, we recommend that you specify the ARNs of only the Lambda functions that you want shared. See the following example.

```
{
  "Sid": "Invoke",
  "Effect": "Allow",
  "Action": [
      "lambda:InvokeFunction"
  ],
  "Resource": [
    "LambdaFunction1ARN",
    "LambdaFunction2ARN"
  ]
 }
```

8. Choose **Save Changes**.

If your IAM policy for dashboard sharing already includes that permission with * as the resource, we strongly recommend that you change the policy and specify only the ARNs of the Lambda functions that you want shared. For example, if your `Resource` section for these permissions was the following:

```
"Resource": "*"
```

Change the policy to specify only the ARNs of the custom widgets that you want shared, as in the following example:

```
"Resource": [
  "LambdaFunction1ARN",
  "LambdaFunction2ARN"
]
```

Using live data in CloudWatch dashboards

You can choose whether your metric widgets display *live data*. Live data is data published within the last minute that has not been fully aggregated.

- If live data is turned **off**, only data points with an aggregation period of at least one minute in the past are shown. For example, when using 5-minute periods, the data point for 12:35 would be aggregated from 12:35 to 12:40, and displayed at 12:41.
- If live data is turned **on**, the most recent data point is shown as soon as any data is published in the corresponding aggregation interval. Each time you refresh the display, the most recent data point may change as new data within that aggregation period is published. If you use a cumulative statistic such as **Sum** or **Sample Count**, using live data may result in a dip at the end of your graph.

You can choose to enable live data for a whole dashboard, or for individual widgets on the dashboard.

Using live data for a CloudWatch dashboard

To choose whether to use live data on your entire dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. To permanently turn live data on or off for all widgets on the dashboard, do the following:
   a. Choose **Actions**, **Settings**, **Bulk update live data.**
   b. Choose **Live Data on** or **Live Data off**, and choose **Set**.
4. To temporarily override the live data settings of each widget, choose **Actions**. Then, under **Overrides**, next to **Live data**, do one of the following:
   - Choose **On** to temporarily turn on live data for all widgets.
   - Choose **Off** to temporarily turn off live data for all widgets.
   - Choose **Do not override** to preserve each widget's live data setting.

Using live data for a CloudWatch dashboard widget

To choose whether to use live data on a single widget

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. Select a widget, and choose **Actions**, **Edit**.
4. Choose the **Graph options** tab.
5. Select or clear the check box under **Live Data**.

Viewing an animated CloudWatch dashboard

You can view an animated dashboard that replays CloudWatch metric data that was captured over time. This can help you see trends, make presentations, or analyze issues after they occur.

Animated widgets in the dashboard include line widgets, stacked area widgets, number widgets, and metrics explorer widgets. Pie graphs, bar charts, text widgets, and logs widgets are displayed in the dashboard but are not animated.

To view an animated dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose the name of the dashboard.
4. Choose **Actions**, **Replay dashboard**.
5. (Optional) By default, when you start the animation, it appears as a sliding window. If you want the animation to appear as a point-by-point animation instead, choose the magnifying glass icon while the animation is paused and reset the zoom.
6. To start the animation, choose the Play button. You can also choose the back and forward buttons to move to other points in time.
7. (Optional) To change the time window for the animation, choose the calendar and select the time period.
8. To change the speed of the animation, choose **Auto speed** and select the new speed.
9. When you are finished, choose **Exit animate**.

Adding a CloudWatch dashboard to your favorites list

In the CloudWatch console, you can add dashboards, alarms, and log groups to a favorites list. You can access the favorites list from the navigation pane. The following procedure describes how to add a dashboard to the favorites list.

To add a dashboard to the favorites list

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. From the list of dashboards, select the star symbol next to the name of the dashboard that you want to favorite.
   - (Optional) You also can favorite a dashboard by selecting a dashboard from the list and choosing the star symbol next to the dashboard name.
4. To access the favorites list, choose **Favorites and recents** in the navigation pane. The menu contains two columns. One contains your favorite dashboards, alarms, and log groups, and the other column contains the dashboards, alarms, and log groups that you recently visited.

> **Tip**
>
> You can favorite dashboards, as well as alarms and log groups, from the ***Favorites and recents*** menu in the CloudWatch console navigation pane. Under the ***Recently visited*** column, hover over the dashboard that you want to favorite, and choose the star symbol next to it.

Changing the period override setting or refresh interval for the CloudWatch dashboard

You can specify how the period setting of graphs added to this dashboard are retained or modified.

When an auto period or persisted time range is applied to a widget, the overall time range of the graph can affect the periods that you have set.

- If the time range is one day or less, period settings are not changed.

- If the time range is between one day and three days, periods set to below five minutes are changed to 5 minutes.
- If the time range is more than three days, periods set to below one hour are changed to one hour.

The following steps explain how to use the console to change the period override options. You can also change them by using the `period0verride` field in the JSON structure of the dashboard. For more information, see Dashboard Body Overall Structure.

To change the period override options

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Actions**.
3. Under **Period override**, choose one of the following:
   - Choose **Auto** to have the period of the metrics on each graph automatically adapt to the dashboard's time range.
   - Choose **Do not override** to make sure that the period setting of each graph is always obeyed.
   - Choose one of the other options to cause graphs added to the dashboard to always adapt that chosen time as their period setting.

   The **Period override** always reverts to **Auto** when the dashboard is closed or the browser is refreshed. Different settings for **Period override** can't be saved.

You can change how often the data on your CloudWatch dashboard is refreshed.

To change the dashboard refresh interval

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. On the **Refresh options** menu (upper-right corner), choose **10 Seconds**, **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Changing the time range or time zone format of a CloudWatch dashboard

You can change the time range to display dashboard data over minutes, hours, days, or weeks. You also can change the time zone format to display dashboard data in UTC or local time. Local time is the time zone that's specified in your computer's operating system.

Note

If you create a dashboard with graphs that contain 100 or more high-resolution metrics, we recommend that you don't set the time range to longer than 1 hour. For more information, see High-resolution metrics.

Note

If the time range of a dashboard is shorter than the period used for a widget on the dashboard, the following happens:

- The widget is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
- The start time of the period for this data point is adjusted backwards to make sure that at least one data point can be displayed.

New console

To change the dashboard time range

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
   - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, or **1w**).
   - Alternatively, you can choose one of the following custom time range options:

- Choose **Custom**, and then choose the **Relative** tab. Choose a time range from 1 minute to 15 months.
- Choose **Custom**, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

> **Tip**
>
> If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the dashboard, choose the time zone dropdown, which shows the current setting, such as **UTC** or **Local time**.
4. Select **UTC** or **Local time** from the dropdown.
5. Choose **Apply**, if prompted.

Old console

To change the dashboard time range

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. From the dashboard screen, do one of the following:
   - In the upper area of the dashboard, select one of the predefined time ranges. These span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**).
   - Alternatively, you can choose one of the following custom time range options:
     - Choose the **custom** dropdown, and then choose the **Relative** tab. Select one of the predefined ranges, which span from 1 minute to 15 months.

- Choose the **custom** dropdown, and then choose the **Absolute** tab. Use the calendar or text fields to specify your time range.

> **Tip**
>
> If the aggregation period is set to **Auto** when you change the time range of a graph, CloudWatch might change the period. To set the period manually, choose the **Actions** dropdown, and then choose **Period override**.

To change the dashboard time zone format

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**, and then choose a dashboard.
3. In the upper-right corner of the dashboard screen, choose the **Custom** dropdown.
4. In the upper-right corner of the box that appears, select **UTC** or **Local timezone** from the dropdown.

Dashboard Body Structure and Syntax

Contents

- Overall Structure
- Widgets Array Structure
- Variables Array Structure
- Properties of a Log Widget Object
- Properties of a Metric Widget Object
- Metric Widget: Format for Each Metric in the Array
- Properties of a Metrics Explorer Widget Object
- Properties of an Alarm Status Widget Object
- Properties of a Chart Widget Object

Overall Structure

A `DashboardBody` is a string in JSON format. It can include an array of between 0 and 500 widget objects, as well as a few other parameters. The dashboard must include a `widgets` array, but that array can be empty.

The following is an example of this structure with one metric widget and one text widget, a time range starting six hours before the current time, and each graph's period setting always being obeyed.

```
{
   "start": "-PT6H",
   "periodOverride": "inherit",
   "widgets": [
      {
         "type":"metric",
         "x":0,
         "y":0,
         "width":12,
         "height":6,
         "properties":{
            "metrics":[
               [
                  "AWS/EC2",
                  "CPUUtilization",
                  "InstanceId",
                  "i-012345"
               ]
            ],
            "period":300,
            "stat":"Average",
            "region":"us-east-1",
            "title":"EC2 Instance CPU",
            "liveData": false,
            "legend": {
                "position": "right"
              }
         }
      },
      {
         "type":"text",
         "x":0,
         "y":7,
```

```
         "width":3,
         "height":3,
         "properties":{
            "markdown":"Hello world"
         }
      }
   ]
}
```

The next example displays three Lambda per-function metrics, and uses a dashboard variable so that dashboard users can switch between different Lambda function names and see all three metrics for each function. This helps you create a single flexible dashboard that can display key metrics for different resources. In this example the functions are discovered by a metric search query, so the dashboard automatically discovers new Lambda functions when they are created.

```
{
    "widgets": [{
            "height": 6,
            "width": 6,
            "y": 0,
            "x": 0,
            "type": "metric",
            "properties": {
                "view": "timeSeries",
                "stacked": false,
                "metrics": [
                    [ "AWS/Lambda", "Invocations", "FunctionName", "my-function-name" ]
                ],

                "region": "us-east-1",
                "liveData": true
            }
        },
        {
            "height": 12,
            "width": 12,
            "y": 0,
            "x": 6,
            "type": "metric",
            "properties": {
                "view": "timeSeries",
                "stacked": false,
                "metrics": [
```

```
                    [ "AWS/Lambda", "Errors", "FunctionName", "my-function-name" ]
                ],

                "region": "us-east-1",
                "liveData": true
            }
        },

        {
            "height": 3,
            "width": 6,
            "y": 0,
            "x": 18,
            "type": "metric",
            "properties": {
                "view": "timeSeries",
                "stacked": false,
                "metrics": [
                    [ "AWS/Lambda", "Duration", "FunctionName", "my-function-name" ]
                ],

                "region": "us-east-1",
                "liveData": true
            }
        }
    ],
    "variables": [{
        "type": "property",
        "property": "FunctionName",
        "inputType": "select",
        "id": "LambdaFunction_Variable",
        "label": "Function",
        "visible": true,
        "search": "{AWS/Lambda,FunctionName} MetricName=\"Duration\"",
        "populateFrom": "FunctionName"
    }]
}
```

The next example has two widgets. The first includes two metrics and a math expression that sums their total. The second widget is a search expression that displays the `CPUUtilization` for all EC2 instances in the Region.

```
{
   "start": "-PT9H",
   "periodOverride": "inherit",
   "widgets": [
      {
         "type":"metric",
         "x":0,
         "y":0,
         "width":12,
         "height":6,
         "properties":{
            "metrics":[
               [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-123",{ "id": "m1" } ],
               [ ".", ".", ".", "i-abc", { "id": "m2" } ],
               [ { "expression": "SUM(METRICS())", "label": "Sum of DiskReadbytes",
 "id": "e3" } ]
            ],
            "view": "timeSeries",
            "stacked": false,
            "period":300,
            "stat":"Average",
            "region":"us-east-1",
            "title":"EC2 Instance CPU"
         }
      },
      {
         "type":"metric",
         "x":0,
         "y":0,
         "width":18,
         "height":9,
         "properties":{
            "metrics":[
               [ { "expression": "SEARCH('{AWS/EC2,InstanceId} MetricName=
\"CPUUtilization\"', 'Average', 300)", "id": "e1" } ]
            ],
            "view": "timeSeries",
            "stacked": false,
            "region":"us-east-1",
            "title":"EC2 Instance CPU"
         }
      }
   ]
}
```

The rest of this section includes examples illustrating each part of the `DashboardBody` syntax. For more examples showing the entire command syntax, see PutDashboard in the Amazon CloudWatch API Reference.

The top level of the JSON object can include the following properties.

widgets

The list of widgets in the dashboard. For more information, see Widgets Array Structure.

Required: Yes

variables

The array of dashboard variable objects used in the dashboard. For more information about the fields that you can use in each dashboard variable object, see Variables Array Structure.

For more information about dashboard variables, see Create flexible dashboards with dashboard variables.

If you include a `variables` array, it can contain between 0 and 25 variable objects.

Required: No

end

The end of the time range to use for each widget on the dashboard when the dashboard loads. If you specify a value for `end`, you must also specify a value for `start`. For each of these values, specify an absolute time in the ISO 8601 format. For example, `2018-12-17T06:00:00.000Z`.

Type: String

Required: No

start

The start of the time range to use for each widget on the dashboard.

You can specify `start` without specifying `end` to specify a relative time range that ends with the current time. In this case, the value of `start` must begin with `-PT` if you specify a time range in minutes or hours, and must begin with `-P` if you specify a time range in days, weeks, or months. You can then use M, H, D, W and M as abbreviations for minutes, hours, days, weeks

and months. For example, `-PT5M` shows the last 5 minutes, `-PT8H` shows the last 8 hours, and `-P3M` shows the last three months.

You can also use `start` along with an `end` field, to specify an absolute time range. When specifying an absolute time range, use the ISO 8601 format. For example, `2018-12-17T06:00:00.000Z`.

If you omit `start`, the dashboard shows the default time range when it loads.

Type: String

Required: No

periodOverride

Use this field to specify the period for the graphs when the dashboard loads. Specifying `auto` causes the period of all graphs on the dashboard to automatically adapt to the time range of the dashboard. Specifying `inherit` ensures that the period set for each graph is always obeyed.

Valid Values: auto | inherit

Type: String

Required: No

Widgets Array Structure

Each widget of any type can have the following properties.

type

The type of widget.

Valid Values: `metric` | `text` | `log` | `alarm` | `explorer` | `chart`

Type: String

Required: Yes

x

The horizontal position of the widget on the 24-column dashboard grid. The default is the next available position.

Valid Values: 0–23

Type: Integer

Required: Yes, if `y` is specified. Otherwise not required.

y

The vertical position of the widget on the 24-column dashboard grid. The default is the next available position.

Valid Values: Any integer, 0 or higher.

Type: Integer

Required: Yes, if `x` is specified. Otherwise not required.

width

The width of the widget in grid units (in a 24-column grid). The default is 6.

Valid Values: 1–24

Type: Integer

Required: No

height

The height of the widget in grid units. The default is 6.

Valid Values: 1–1000

Type: Integer

Required: No

properties

The detailed properties of the widget, which differ depending on the widget type. For more information about the format of `properties`, see Properties of a Metric Widget Object, Properties of a Chart Widget Object, or Properties of a Text Widget Object.

Type: Object

Required: Yes

Variables Array Structure

Each dashboard variable in the array can have the following properties.

type

The type of dashboard variable. CloudWatch supports two types, *property variables* and *pattern variables*.

Property variables change the values of all instances of a property in all widgets in the dashboard. A property can be either of the following:

- Any JSON property in the widget definitions, such as `region`.
- Any dimension name for a metric, such as `InstanceId` or `FunctionName`.

A pattern variable changes a regular expression pattern across the dashboard JSON. Use it when you need to change just part of a JSON property value, or even a JSON property.

Property variables apply to most use cases and are less complex to set up.

Valid Values: `property` | `pattern`

Type: String

Required: Yes

inputType

Determines how the dashboard user inputs the value for the variable.

- Specify `input` to use a text box that the user can enter values into.
- Specify `select` to use a dropdown set of values that you define, or a dropdown list of values found by a metric search query.
- Specify `radio` to use a set of radio buttons, with values that you define or that you find with a metric search query.

Valid Values: `input` | `select` | `radio`

Type: String

Required: Yes

values

If your `inputType` is `select` or `radio` and you want to define the possible variable values instead of using a metric query search, specify those values here.

`values` is an array where each object in the array contains a required *value* and an optional *label*. Each *value* can be a string, number, or Boolean, and each label must be a string. Each of these values and labels can be as many as 255 characters.

If you specify a `values` array it must include at least one item, and can include as many as 500.

For example, the following creates a list of three possible Regions to use as the values for a variable.

```
"values": [
    { "label": "US East (IAD)", "value": "us-east-1" },
    { "label": "US West (SFO)", "value": "us-west-1" },
    { "label": "EU (DUB)", "value": "eu-west-1" }
]
```

Type: Array

Required: Yes if `inputType` is `select` or `radio` and you are not using a metric search query to populate the values.

id

An Id for this variable. It can be up to 32 characters, and valid characters are `0-9A-Za-z-_`

Type: String

Required: Yes

label

A label to display for the input field. It can have as many as 30 characters.

If you omit this field for a property variable, the property name is displayed as the label. If you omit it for a pattern variable, then *`pattern_1`*, *`pattern_2`*, `...` is used for the label.

Type: String

Required: No

defaultValue

The default value for the variable, when the dashboard is first opened.

- If `inputType` is `input`, you specify the `defaultValue` here manually and use as many as 255 characters
- If `inputType` is `select` or `radio`, you must specify a valid possible value that you specified in the `values` array or that was retrieved in the metric query search.

Valid Values: Any valid value for this variable

Type: String, Number, or Boolean, depending on the type value for this variable

Required: No

search

Specify this field to populate your `select` or `radio` input field by using a metric search expression. For the value for this field, specify a namespace, dimension name, and a metric name. The dimension that you specify must be valid for that metric. CloudWatch finds all resources that publish that metric and dimension, and populates the list with them.

For example, specify `"search": "{AWS/EC2,InstanceId} MetricName=\"CPUUtilization\""` to search for Amazon EC2 instances in the account, or specify `"search": "{AWS/Lambda,FunctionName} MetricName=\"Duration\""`, to return the Lambda functions in the account

The `search` string that you specify can be as many as 2048 characters.

> **Note**
>
> If you're using a search expression and you also want to specify a default value, the default that you specify in `defaultValue` is used as long as that default is one of the resources that was retrieved by the search. When using a search expression to populate your input field, you can also specify the special value `__FIRST` for `defaultValue`, to have the default value be the first value returned from the search. (The special value includes two underscores and then FIRST) Values returned by the search are always sorted alphabetically.

Type: String

Required: Yes if `inputType` is `select` or `radio` and you are not specifying `values`.

populateFrom

If you are using the `search` field to populate your input field by using a search expression, specify this field with the name of a dimension that the search will retrieve.

For example, if your `search` value is `"search": "{AWS/EC2,InstanceId} MetricName=\"CPUUtilization\""`, then you can specify `InstanceId` for `populateFrom`.

Type: String

Required: Yes if `inputType` is `select` or `radio` and you are not specifying `values`.

visible

Specified whether the input label and field are visible on the dashboard. If you omit this, the default of `true` is used.

Using `false` saves some room on the dashboard, but requires the user to change the dashboard URL to change the variable values.

Type: Boolean

Required: No

Variable examples

The following example uses a property variable for changing the Region of all widgets, using a text input field. When the dashboard is first opened, the default value of `us-east-1` is used for the variable.

```
"variables": [
     {
        "type": "property",
        "property": "region",
        "inputType": "input",
        "id": "region",
        "label": "Region",
        "defaultValue": "us-east-1",
        "visible": true
    }
],
```

The following example uses a pattern variable for changing the Region of all widgets, in the case where sometimes the Region is set in the middle of a string such as an ARN.

```
"variables": [
    {
        "type": "pattern",
        "pattern": "us-east-1",
        "inputType": "input",
        "id": "region",
        "label": "Region",
        "defaultValue": "us-east-1",
        "visible": true
    }
],
```

The following example generates a Lambda function variable, with a radio button for each function. Functions are discovered by a metric query search.

```
"variables": [
    {
        "type": "pattern",
        "pattern": "originalFuncNameInDashboard",
        "inputType": "radio",
        "id": "functionName",
        "label": "Function",
        "visible": true,
        "search": "{AWS/Lambda,FunctionName} MetricName=\"Duration\"",
        "populateFrom": "FunctionName",
        "defaultValue": "__FIRST"
    }
],
```

The following sample displays how to specify multiple variables in a dashboard, and demonstrates several types of variables.

```
"variables": [{
        "type": "property",
        "property": "region",
        "inputType": "select",
        "id": "unique_id_1",
        "label": "Region",
        "defaultValue": "us-east-1",
```

```
        "visible": true,
        "values": [{
                "label": "IAD",
                "value": "us-east-1"
            },
            {
                "label": "CMH",
                "value": "us-east-2"
            },
            {
                "label": "NRT",
                "value": "ap-northeast-1"
            }
        ]
    },
    {
        "type": "property",
        "property": "FunctionName",
        "inputType": "select",
        "id": "unique_id_2",
        "label": "Function",
        "visible": true,
        "values": [{
                "value": "my-FunctionName-1"
            },
            {
                "value": "my-FunctionName-2"
            },
            {
                "value": "my-FunctionName-3"
            }
        ]
    },
    {
        "type": "property",
        "property": "accountId",
        "inputType": "radio",
        "id": "unique_id_3",
        "defaultValue": "111122223333",
        "visible": true,
        "values": [{
                "label": "IAD Account",
                "value": "111122223333"
            },
```

```
            {
                "label": "CMH Account",
                "value": "123456789012"
            },
            {
                "label": "NRT Account",
                "value": "000000000000"
            }
        ]
    }
]
```

Properties of a Text Widget Object

A widget of type `text` can have one or two parameters in the `properties` section. The `markdown` field is required, and the `background` field is optional.

For more information about the style of markdown supported in CloudWatch text widgets, see Using Markdown in the Console.

markdown

The text to be displayed by the widget. Use this parameter only for text widgets.

Type: String

Required: Yes (when the widget `type` is `text`).

background

Specifies whether the text widget has a solid or transparent background. The value `transparent` makes the widget transparent. The value `solid` is the default.

Type: String

Required: No

```
{
   "widgets":[
      {
         "type":"text",
         "x":0,
         "y":7,
```

```
         "width":3,
         "height":3,
         "properties":{
            "markdown":"Hello world",
            "background": "transparent"
         }
      }
   ]
}
```

Properties of a Log Widget Object

A widget of type `log` represents the results of a CloudWatch Logs Insights query. For more information, see Analyzing Log Data with CloudWatch Logs Insights.

A `log` widget can include the following fields in its `properties` field.

accountId

The account ID of the AWS account containing the logs, if this is a cross-account query.

Type: String

Required: No

region

The Region of the logs query.

Type: String

Required: Yes

title

The title text to be displayed by the widget.

Type: String

Required: No

query

Contains the CloudWatch Logs Insights query function.

Type: String

Required: Yes (when the widget `type` is `log`).

The `query` string starts with the names of the log groups that are to be queried. You must prepend each log group name with `SOURCE`. Separate multiple log groups with a pipe character (|).

Add another pipe character after the list of log groups, and then specify the query syntax. Separate each line in the query syntax with `\n|`

For example, the following line represents a query of two log groups, `service_log1` and `service_log2`. The query displays canaries that have faults.

```
"query": "SOURCE 'service_log1' | SOURCE 'service_log2' |filter Fault > 0\n| fields
 Fault.Message\n| stats count(*) by Canary.Name, Fault.Message"
```

view

Specifies how the query results are displayed. Specify `table` to view the results as a table. Specify `timeSeries` to display this metric as a line graph. Specify `bar` to display it as a bar graph. Specify `pie` to display it as a pie graph.

If you omit this parameter, the results are displayed as a table.

Type: String

Required: No (when the widget `type` is `log`).

```
{
    "widgets": [
        {
            "type": "log",
            "x": 12,
            "y": 24,
            "width": 12,
            "height": 6,
            "properties": {
                "region": "us-east-1",
                "title": "Errors (Application Log)",
                "query": "SOURCE 'application1.log' | SOURCE 'application2.log' |
 filter @message like \"[ERROR]\"\n| parse \"Error for [*] [*] due to: *\" canaryName1,
 canaryId1, cause1\n| parse \"Executor canary [*] *\" canaryName2, cause2\n| fields
 coalesce(cause1, cause2) as cause\n| fields coalesce(canaryName1, canaryName2)
```

```
 as canaryName\n| fields ispresent(cause) as isP\n| filter isP\n| stats count() as
 errCount by canaryName, substr(cause, 0, 130)\n| sort errCount DESC",
                "view": "table"
            }
        }
    ]
}
```

Properties of a Metric Widget Object

A widget of type `metric` can have the following fields within `properties`:

accountId

Specifies the AWS account ID where all metrics in this widget will come from. This is useful for cross-account dashboards that include widgets from multiple accounts. For more information, see Cross-Account Cross-Region CloudWatch Console.

If you omit this, the current account is used as the default. Use this parameter only for metric widgets.

You can also use an `accountId` field within each metric in the array of `metrics` to create a single widget which includes metrics from multiple accounts.

Type: String

Required: No

annotations

To include an alarm or annotation in the widget, specify an `annotations` array. For more information about the format, see Dashboard Widget Object: Annotation Properties. Use this parameter only for metric widgets.

Type: Object

Required: An alarm annotation is required only when the widget `type` is `metric` and `metrics` is not specified. A horizontal or vertical annotation is not required.

liveData

Specify `true` to display *live data* in the widget. Live data is data published within the last minute that has not been fully aggregated. For more information, see Use Live Data.

Type: Boolean

Required: No

legend

Specify `legend` to determine where the legend for the lines on the graph is displayed. The `legend` field contains another field called `position`. Possible values for `position` are `right`, `bottom`, and `hidden`.

For example, the following causes the legend to appear on the right in the graph.

```
"legend": {
     "position": "right"
}
```

Type: Object

Required: No

metrics

Specify a `metrics` array to include one or more metrics (without alarms), a Metrics Insights query, math expressions, or search expressions. One `metrics` array can include 0–500 metrics and expressions. Use this parameter only for metric widgets. For more information about the format of `metrics`, see Metric Widget: Format for Each Metric in the Array.

One metrics array can include no more than one Metrics Insights query.

A single `expression` field can't include both a Metrics Insights query and a math expression. However, you can use the returned results of a Metrics Insights query from one expression as input in a math expression in a different expression in the array.

Type: Array of arrays

Required: Yes, when the widget `type` is `metric` and `annotations` is not specified.

period

The default period, in seconds, for all metrics in this widget. The period is the length of time represented by one data point on the graph. This default can be overridden within each metric definition. Use this parameter only for metric widgets. The default is 300.

Valid Values: Any multiple of 60, with 60 as the minimum.

Type: Integer

Required: No

region

The region of the metric.

Type: String

Required: Yes

sparkline

Specify `true` to display the sparkline feature under the number widget. Specify `false` to display the number widget by itself. This parameter is ignored if view is not `singleValue`. Use this parameter only for metric widgets.

Type: Boolean

Required: No

stacked

Specify `true` to display the graph as a stacked line, or `false` to display as separate lines. This parameter is ignored if `view` is `singleValue`. Use this parameter only for metric widgets.

Type: Boolean

Required: No

stat

The default statistic to be displayed for each metric in the array. This default can be overridden within the definition of each individual metric in the `metrics` array. Use this parameter only for metric widgets.

Valid Values: `SampleCount` | `Average` | Sum | `Minimum` | `Maximum` | p*??*

Type: String that is a valid CloudWatch statistic.

Required: No

table

Include a `table` section to include data table-related changes in the widget. For more information about the format, see Dashboard Widget Object: Table Properties.

Type: Object

Required: No

timezone

The time zone to use for displaying the times in the graph. The format is + or - followed by four digits. The first two digits indicate the number of hours ahead or behind of UTC, and the final two digits are the number of minutes. For example, `+0130` indicates a time 1 hour and 30 minutes ahead of UTC. The default is `+0000`.

Type: String

Required: No

title

The title to be displayed for the graph or number. Use this parameter only for metric widgets.

Type: String

Required: No

view

Specify `timeSeries` to display this metric as a line or stacked area graph. Specify `singleValue` to display this metric as a number graph. Specify `gauge` to display this metric as a gauge graph. Specify `bar` to display this metric as a bar graph. Specify `pie` to display this metric as a pie graph.

> **Note**
>
> If you specify `gauge`, you must set a value for `min` and `max` on the left side of `yAxis`.

Valid Values: `timeSeries` | `singleValue` | `gauge` | `bar` | `pie` | `table`

Type: String

Required: No

yAxis

The minimum and maximum values for the left and right side of a graph's Y-axis. This property applies to all graphed metrics, except for specific metrics that override this setting. For more information, see Dashboard Widget Object: yAxis Properties Format.

Type: yAxis object

Required: No

Example: Stacked area and gauge widgets

```
{
    "widgets": [
{
   "type":"metric",
   "x":0,
   "y":0,
   "width":12,
   "height":6,
   "properties":{
      "metrics":[
         [
            "AWS/EC2",
            "CPUUtilization",
            "InstanceId",
            "i-012345"
         ],
         [
            "AWS/EC2",
            "NetworkIn",
            "InstanceId",
            "i-012345",
            {
               "yAxis":"right",
               "label":"NetworkIn",
               "period":3600,
               "stat":"Maximum"
            }
         ]
      ],
      "period":300,
      "stat":"Average",
```

```
      "region":"us-east-1",
      "timezone":"+0300",
      "title":"EC2 Instance CPU",
      "stacked":true,
      "view":"timeSeries",
      "liveData":false,
      "yAxis":{
         "left":{
            "min":0,
            "max":100
         },
         "right":{
            "min":50
         }
      },
      "annotations":{
         "horizontal":[
            {
               "visible":true,
               "color":"#9467bd",
               "label":"Critical range",
               "value":20,
               "fill":"above",
               "yAxis":"right"
            }
         ]
      }
   }
},
{
   "type": "metric",
   "x": 18,
   "y": 60,
   "width": 6,
   "height": 6,
   "properties": {
      "metrics": [
         [
         "AWSLogsShrinkRay",
         "disk_inodes_used",
         "path",
         "/dev/shm",
         "InstanceId",
         "i-012345",
```

```
         "AutoScalingGroupName",
         "ShrinkRayExecutorResourceStack-Gamma-us-east-1-
ASGuseast1ac48xlargeASGB9B53974-VTYXJUZGUAHV",
         "InstanceType",
         "c4.8xlarge",
         "device",
         "tmpfs",
         "fstype",
         "tmpfs"
         ]
         ],
      "view": "gauge",
      "title": "Disk Inodes Used",
      "region": "us-east-1",
      "yAxis": {
         "left": {
            "min": 0,
            "max": 100
         }
       }
     }
   }
   ]
}
```

Metric Widget: Format for Each Metric in the Array

Each item in the `metrics` array is either a single metric or a math expression or search expression. Each single metric in the `metrics` array has the following format:

```
[ Namespace, MetricName, [{DimensionName,DimensionValue}...] {Rendering Properties
 Object} ]
```

Each expression in the `metrics` array has the following format:

```
[ {"expression" : "expression", ["label" : "label"] , ["id" : "id"] } ]
```

accountId

Specifies the AWS account ID where this metric comes from. This enables you to create a widget that contains metrics from multiple accounts on a cross-account dashboard. For more information, see Cross-Account Cross-Region CloudWatch Console.

If you omit this, the current account is used as the default. Use this parameter only for metric widgets.

Type: String

Required: No

Namespace

The AWS namespace containing the metric. If you have multiple entries in the `metrics` array, for each one after the first you may specify only `"."` to use the same namespace as the previous metric in the array.

Type: String

Required: Yes

MetricName

The name of the CloudWatch metric. If you have multiple entries in the `metrics` array, for each one after the first you may specify only `"."` to use the same metric name as the previous metric in the array.

Type: String

Required: Yes, for a single metric

expression

The Metrics Insights query, math expression, or search expression, if this is an expression instead of a single metric.

In a search expression using double-quotes for an exact match, each double-quote mark must be escaped with a backslash.

For more information about Metrics Insights query syntax, see Metrics Insights query components and syntax.

For more information about math expressions or search expressions, see Using Metric Math or Using Search Expressions in Graphs in the Amazon CloudWatch User Guide.

Type: String

Required: Yes, for an expression

Example of an expression for a Metrics Insights query:

```
[ { "expression": "SELECT MAX(CPUUtilization) FROM SCHEMA(\"AWS/EC2\", InstanceId)
 GROUP BY InstanceId LIMIT 10", "label": "View the 10 max CPU Utilization", "id":
 "q1" } ]
```

DimensionName

The name of a dimension to further refine what data is shown. If you have multiple entries in the `metrics` array, for each one after the first you may specify only `"."` to use the same dimension name as in the corresponding dimension specified in the previous metric in the array. You may specify 0 dimensions for a metric, or up to as many dimensions as the metric support.

Type: String

Required: No

DimensionValue

The value to use for that dimension for the metric. Required if there is a corresponding dimension name.

Type: String

Required: No

id

The Id of this time series. This Id can be used as part of a math expression. The Id must start with a lowercase letter.

Type: String

Required: No

label

The label to display in the graph to represent this time series.

Type: String

Required: No

region

The region of the metric. Use this parameter only for metric widgets. If you omit this, the current Region is used as the default.

Type: String

Required: No

Rendering Properties Object

Specifies rendering properties to be used for this particular metric, overriding the values specified for the overall widget. For more information about the format, see Dashboard Widget Object: Rendering Properties Object Format.

Type: Metric Render Properties Object

Required: No

```
// The simplest example, a metric with no dimensions
        [ "AWS/EC2", "CPUUtilization" ]

 // A metric with a single dimension
        [ "AWS/EC2", "CPUUtilization", "InstanceId", "i-012345" ]

 // A metric with a single dimension and rendering properties
        [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-xyz", { "yAxis": "right"} ]

 // The following example graphs the DiskReadBytes metric for three instances.
        [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-xyz" ],
        [ ".", ".", ".", "i-abc" ],
        [ ".", ".", ".", "i-123" ]

 // The following example includes two metrics and a math expression to sum them.
        [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-123",{ "id": "m1" } ],
        [ ".", ".", ".", "i-abc", { "id": "m2" } ],
        [ { "expression": "SUM(METRICS())", "label": "Sum of DiskReadbytes", "id":
"e3" } ]

  // The following example is a search expression showing the EC2 CPUUtilization for
each instance in the Region.
        [ { "expression": "SEARCH('{AWS/EC2,InstanceId} MetricName=\"CPUUtilization\"',
'Average', 300)", "id": "e1" } ],
```

Topics

- Dashboard Widget Object: Rendering Properties Object Format
- Dashboard Widget Object: Annotation Properties

- Dashboard Widget Object: yAxis Properties Format
- Dashboard Widget Object: Table Properties

Dashboard Widget Object: Rendering Properties Object Format

Each metric in the `metrics` array can optionally have custom rendering properties that override the default rendering properties specified in the `yAxis` parameter of the `widget` object. This section describes the format for those per-metric custom rendering properties.

color

The six-digit HTML hex color code to be used for this metric.

Type: String

Required: No

label

The label to display for this metric in the graph legend. If this is not specified, the metric is given an autogenerated label that distinguishes it from the other metrics in the widget.

Type: String

Required: No

period

The period for this metric, in seconds. The period is the length of time represented by one data point on the graph.

Valid Values: A multiple of 60, with a minimum of 60.

Type: Integer

Required: No

stat

The statistic for this metric, if it is to be different than the statistic used for the other metrics in the array. By default CloudWatch uses *Average* if you don't specify a statistic at array or at metric level.

Valid Values: `SampleCount` | `Average` | `Sum` | `Minimum` | `Maximum` | p*??*

Type: String that is a valid CloudWatch statistic.

Required: No

visible

Set this to `true` to have the metric appear in the graph, or `false` to have it be hidden. The default is `true`.

Type: Boolean

Required: No

yAxis

Where on the graph to display the y-axis for this metric. The default is `left`.

Valid Values: `left` | `right`

Type: String

Required: No

```
// The third metric has its own rendering properties, overriding those of the rest of
the widget.
       [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-xyz" ],
       [ ".", ".", ".", "i-abc" ],
       [ ".", ".", ".", "i-123", { "label":"Instance i-123", "yAxis": "right"}  ]
```

Dashboard Widget Object: Annotation Properties

Annotations include alarms, horizontal annotations, and vertical annotations. A single metric widget can have up to one alarm, or it can have one or more horizontal or vertical annotations. A single widget can't have both an alarm and horizontal or vertical annotations.

Alarm Annotations

If you specify an alarm annotation, you cannot also specify a `metrics` array in the same widget.

alarms

The Amazon Resource Name (ARN) of the alarm.

Type: Array of strings. There can be 0–1 strings in the array.

Required: Only if no metrics are listed.

```
"annotations": {
   "alarms": [ "arn1" ]
}
```

Horizontal Annotations

horizontal

An array of horizontal annotations. Horizontal annotations have several options for fill shading, including shading above the annotation line, shading below the annotation line, and "band" shading that appears between two linked annotation lines as part of a single band annotation. Each horizontal annotation in the array that is a single annotation, instead of a band annotation, has the following format:

```
{value, label, color, fill, yAxis, visible}
```

Each horizontal annotation that is a band annotation has the following format:

```
[ {value, label, color, yAxis, visible}, {value, label} ]
```

value

The metric value in the graph where the horizontal annotation line is to appear. On a band shading annotation, the two values for Value define the upper and lower edges of the band.

On a graph with horizontal annotations, the graph is scaled so that all visible horizontal annotations appear on the graph.

Type: Float

Required: Yes

label

A string that appears on the graph next to the annotation.

Type: String

Required: No

color

The six-digit HTML hex color code to be used for the annotation. This color is used for both the annotation line and the fill shading.

Type: String

Required: No

fill

How to use fill shading with the annotation. Valid values are `above` for shading above the annotation, `below` for shading below the annotation, and `none` for no shading. If `fill` is omitted, there is no shading.

The exception is an annotation with band shading. These annotations always have shading between the two values, and any value for `fill` is ignored.

Type: String

Required: No

visible

Set this to `true` to have the annotation appear in the graph, or `false` to have it be hidden. The default is `true`.

Type: Boolean

Required: No

yAxis

If the graph includes multiple metrics, specifies whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, . Valid values are `right` and `left`.

Type: String

Required: No

```
// A single horizontal annotation with fill shading above the annotation line, based on
 the metric associated with the right Y-axis

"annotations": {
     "horizontal": [
         {
              "visible":true,
              "color":"#9467bd",
              "label":"Critical range",
              "value":20,
              "fill":"above",
              "yAxis":"right"
         }
    ]
}

// A band annotation. Each value has a label, but other parameters for the band are
 specified only with the first number

"annotations": {
    "horizontal": [
        [
            {
                "label": "Band top",
                "value": 200,
                "color": "#9467bd",
                "visible": true,
                "yAxis": "right"
            },
            {
                "value": 95.5,
                "label": "Band bottom"
            }
        ]
    ]
}

// Three annotations on a graph. The first one is a band annotation. The final one is
 hidden.
```

```
"annotations": {
    "horizontal": [
        [
            {
                "label": "Band top",
                "value": 200,
                "color": "#9467bd",
                "visible": true,
                "yAxis": "right"
            },
            {
                "value": 95.5,
                "label": "Band bottom"
            }
        ],
        {
            "visible": true,
            "color": "#9467bd",
            "label": "Label for this annotation",
            "value": 20,
            "fill": "below",
            "yAxis": "right"
        },
        {
            "visible": false,
            "color": "#aaa",
            "label": "Hidden annotation",
            "value": 150
        }
    ]
}
```

Vertical Annotations

vertical

An array of vertical annotations. For each vertical annotation, you can choose to have fill shading before the annotation, after it, or between two vertical lines that are linked as a single band annotation. Each vertical annotation in the array that is a single annotation, instead of a band annotation, has the following format:

```
{value, label, color, fill, visible}
```

Each vertical annotation that is a band annotation has the following format:

```
[ {value, label, color, visible}, {value, label} ]
```

value

The date and time in the graph where the vertical annotation line is to appear. On a band shading annotation, the two values for Value define the beginning and ending edges of the band.

On a graph with vertical annotations, the graph is scaled so that all visible vertical annotations appear on the graph.

This is defined as a string in ISO 8601 format. For more information, see ISO 8601.

Type: String

Required: Yes

label

A string that appears on the graph next to the annotation.

Type: String

Required: No

color

The six-digit HTML hex color code to be used for the annotation. This color is used for both the annotation line and the fill shading.

Type: String

Required: No

fill

How to use fill shading with the annotation. Valid values are `before` for shading before the annotation, `after` for shading after the annotation, and `none` for no shading. If `fill` is omitted, there is no shading.

The exception is an annotation with band shading. These annotations always have shading between the two values, and any value for `fill` is ignored.

Type: String

Required: No

visible

Set this to `true` to have the annotation appear in the graph, or `false` to have it be hidden. The default is `true`.

Type: Boolean

Required: No

```
// A single vertical annotation with fill shading after the annotation line

"annotations": {
    "vertical": [
        {
            "visible": true,
            "color": "#9467bd",
            "label": "Bug fix deployed",
            "value": "2018-08-28T15:25:26Z",
            "fill": "after"
        }
    ]
}


// A band vertical annotation. Each annotation line has a label, but other parameters
 for the band are specified only with the first value

"annotations": {
    "vertical": [
        [
            {
                "label": "Band start",
                "value": "2018-08-27T15:25:26Z",
                "color": "#9467bd",
                "visible": true
            },
```

```
            {
                "value": "2018-08-28T15:25:26Z",
                "label": "Band end"
            }
        ]
    ]
}
```

Dashboard Widget Object: yAxis Properties Format

Defines the settings for the Y-axis of the graph. The settings include the maximum and minimum, a label for the axis, and whether the axis shows the units. Set this within the `widget` object to affect all metrics in the widget. To override the widget settings for a particular metric, set it for the metric in the `metrics` array.

```
{
  "left": {
    "min": 0,
    "max": 100
  },
  "right": {
    "min": 0
  }
}
```

left

Optional settings for the left Y-axis.

Type: YAxis object

Required: No

right

Optional settings for the right Y-axis.

Type: YAxis object

Required: No

Each of the `left` and `right` objects can include the following parameters:

label

A label for this Y-axis

Type: String

Required: No

min

The minimum value for this Y-axis

Type: Float

Required: No

max

The maximum value for this Y-axis

Type: Float

Required: No

showUnits

Determines whether the units are shown for the metric associated with this axis. The default is true.

Type: Boolean

Required: No

Dashboard Widget Object: Table Properties

If you specify `table` for a metric widget, you can include visualisations related to the visibility of summary columns, datapoint columns, and the table layout. These properties take effect only when the widget view type is `table` and don't alter other view types if included. The `table` property is not required to use a table widget.

layout

Use this field to transform the table such that the data points extend vertically or horizontally. The default is `horizontal`.

Valid Values: `vertical` | `horizontal`

Type: String

Required: No

stickySummary

Set this to `true` to make the summary columns that you include in the table sticky, so you can explore the data columns while always having the summary columns in your viewport. The default is `false`.

The widget label is always sticky, no matter your choice in this field.

Valid Values: `true` | `false`

Type: Boolean

Required: No

showTimeSeriesData

Set this to `false` if you want only the label and summary columns to be displayed, hiding the other columns of data.

The default is `true`.

Valid Values: `true` | `false`

Type: Boolean

Required: No

summaryColumns

Summary columns are a new property introduced with the table widget. These columns are a specific subset of summaries of your current table. For example, the `Sum` summary is a sum of all the rendered datapoints in its respective row. Summary columns are not that same concept as any CloudWatch metric statistic.

The default is `["MIN", "MAX","SUM", "AVG"]`

Valid Values: `"MIN"` | `"MAX"`| `"SUM"`| `"AVG"`

Type: Array

Required: No

For example, the following JSON creates a table displaying the minimum and maximum of each metric in the table.

```
"table": {
    "summaryColumns": ["MIN", "MAX"],
    "layout": "vertical",
    "stickySummary": true,
    "showTimeSeriesData": false
    },
```

Properties of a Metrics Explorer Widget Object

A widget of type `explorer` represents a metrics explorer widget. For more information, see Use Metrics Explorer to Monitor Resources by Their Tags and Properties

You can also add metrics explorer widgets to a dashboard using CloudFormation. For more information, see AWS::CloudWatch::Dashboard.

This widget type can have the following fields within the widget `properties`:

aggregateBy

An object that specifies how to aggregate metrics from multiple resources. The valid values for the `key` field in this object are the keys of tags and resource properties. This object contains the following fields.

- **key**– The tag or resource property key to use for aggregating the metrics.
- **func**– The aggregation function to use. Valid values are `AVG` | `MIN` | `MAX` | `STDDEV` | `SUM`

Type: Object

Required: No

labels

An array of the tags or the resource properties that are used to determine which metrics are displayed in the widget.

If you specify different keys, then only the resources that match all of the key/value pairs are displayed. If you specify multiple values for a single key, then resources that match any of the values for that key are displayed.

- **key**– The tag or resource property to filter on.

  For `key`, all tag keys are valid to be specified. The following EC2 and Lambda resource properties are also valid for `key`:

  - EC2:

    `Architecture`, `Hypervisor`, `CoreCount`, `ImageId`, `InstanceId`, `InstanceLifecycle`, `InstanceType`, `InstanceFamily`, `InstanceSize`, `Affinity`, `AvailabilityZone`, `Tenancy`, `Platform`, `RootDeviceType SecurityGroups`, `State, SubnetId, VirtualizationType, and VpcId`

  - Lambda:

    `FunctionName`, `Runtime`, `Language`, `MemorySize`, `Version`, `SecurityGroupIds`, `SubnetIds`, `SubnetIdCount`, `VpcId`, and `Timeout`

- **value**– (Optional) The value of the tag or resource property to filter on. If this is omitted, metrics corresponding to all values of that tag or resource property are displayed.

  Type: Object

  Required: Yes

metrics

Specify a `metrics` array to include one or more metrics. One `metrics` array can include 1–100 metrics. Each object in the array must contain the following fields.

- **metricName**– The name of the metric.
- **resourceType**– The type of resource publishing the metric, described in the format used by AWS CloudFormation. For example, `AWS::EC2::Instance` or `AWS::Lambda::Function`.

  You must use the same value for `resourceType` for all metrics in the widget.

  For a complete list of valid values, see Valid resourceType Values for a Metric Explorer Widget Object.

- **stat**– The statistic for this metric, if it is to be different than the statistic used for the other metrics in the array. By default CloudWatch uses *Average* if you don't specify a statistic at array or at metric level.

  Valid Values: `SampleCount` | `Average` | `Sum` | `Minimum` | `Maximum` | `p??`

  Type: Array of objects

Required: Yes

period

The default period, in seconds, for all metrics in this widget. The period is the length of time represented by one data point on the graph. The default is 300.

Valid Values: Any multiple of 60, with 60 as the minimum.

Type: Integer

Required: No

splitBy

Specifies how to split the metrics from multiple resources into different lines on a graph, or into different graphs. The valid values are the keys of tags, and the keys of resource properties.

Type: String

Required: No

title

The title to be displayed for the widget. The default is `Explorer`.

Type: String

Required: No

widgetOptions

An object that specifies how the widget appears on the dashboard. It can contain the following fields.

- **legend**–

  Determines where the legend for each graph is displayed. The `legend` field contains another field called `position`. Possible values for `position` are `right`, `bottom`, and `hidden`.

  For example, the following causes the legend to appear on the right in the graph.

  ```
  "legend": {
       "position": "right"
  }
  ```

- **rowsPerPage**–

Specifies how many rows of graphs are displayed per page in the widget.

- **stacked**–

  Specify `true` to display the graph as a stacked area chart, or `false` to display as separate lines.

- **view**–

  Specifies how each graph is displayed. Specify `timeSeries` to display this metric as a line graph. Specify `bar` to display it as a bar graph. Specify `pie` to display it as a pie graph. The default is `timeSeries`.

- **widgetsPerRow**–

  Specifies how many graphs are displayed in each row of the metrics explorer widget.

Type: Object

Required: No

Example

The following example displays three metrics for each of the account's running EC2 instances, with the graphs in the widget split by availability zone. Within each graph, the metrics are aggregated by instance type.

```
{
    "widgets": [
        {
            "type": "explorer",
            "width": 24,
            "height": 15,
            "x": 0,
            "y": 0,
            "properties": {
                "metrics": [
                    {
                        "metricName": "CPUUtilization",
                        "resourceType": "AWS::EC2::Instance",
                        "stat": "Average"
                    },
                    {
```

```
                        "metricName": "NetworkIn",
                        "resourceType": "AWS::EC2::Instance",
                        "stat": "Average"
                    },
                    {
                        "metricName": "NetworkOut",
                        "resourceType": "AWS::EC2::Instance",
                        "stat": "Average"
                    }
                ],
                "aggregateBy": {
                    "key": "InstanceType",
                    "func": "MAX"
                },
                "labels": [
                    {
                        "key": "State",
                        "value": "running"
                    }
                ],
                "widgetOptions": {
                    "legend": {
                        "position": "bottom"
                    },
                    "view": "timeSeries",
                    "rowsPerPage": 8,
                    "widgetsPerRow": 2
                },
                "period": 300,
                "splitBy": "AvailabilityZone",
                "title": "Running EC2 Instances by AZ"
            }
        }
    ]
}
```

Valid resourceType Values for a Metric Explorer Widget Object

The valid values for the `resourceType` field in the `metrics` section of a metrics explorer widget are as follows:

- `AWS::AmazonMQ::Broker`
- `AWS::ApiGateway::RestApi`

- AWS::AppStream::Fleet
- AWS::AppSync::GraphQLApi
- AWS::CloudFront::Distribution
- AWS::CodeBuild::Project
- AWS::Datasync::Agent
- AWS::Datasync::Task
- AWS::DMS::ReplicationInstance
- AWS::DynamoDB::Table
- AWS::EC2::CapacityReservation
- AWS::EC2::Instance
- AWS::EC2::NatGateway
- AWS::EC2::TransitGateway
- AWS::EC2::Volume
- AWS::EC2::VPNConnection
- AWS::ECS::Cluster
- AWS::EFS::FileSystem
- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::ReplicationGroup
- AWS::ElasticBeanstalk::Environment
- AWS::ElasticLoadBalancing::LoadBalancer
- AWS::ElasticLoadBalancingV2::LoadBalancer/ApplicationELB
- AWS::ElasticLoadBalancingV2::LoadBalancer/GatewayELB
- AWS::ElasticLoadBalancingV2::LoadBalancer/NetworkELB
- AWS::ElasticLoadBalancingV2::TargetGroup
- AWS::EMR::Cluster
- AWS::Events::Rule
- AWS::FSx::FileSystem
- AWS::GameLift::Fleet
- AWS::GlobalAccelerator::Accelerator

- AWS::IoT::TopicRule
- AWS::IoT1Click::Device
- AWS::IoTAnalytics::Channel
- AWS::IoTAnalytics::Dataset
- AWS::IoTAnalytics::Datastore
- AWS::IoTAnalytics::Pipeline
- AWS::Kafka::Cluster
- AWS::Kinesis::Stream
- AWS::KinesisAnalytics::Application
- AWS::KinesisFirehose::DeliveryStream
- AWS::KinesisVideo::Stream
- AWS::KMS::Key
- AWS::Lambda::Function
- AWS::Logs::LogGroup
- AWS::MediaPackage::Channel
- AWS::MediaStore::Container
- AWS::OpsWorks::Instance
- AWS::OpsWorks::Layer
- AWS::OpsWorks::Stack
- AWS::QLDB::Ledger
- AWS::RDS::DBInstance
- AWS::Redshift::Cluster
- AWS::RoboMaker::SimulationJob
- AWS::Route53::HealthCheck
- AWS::Route53Resolver::ResolverEndpoint
- AWS::S3::Bucket
- AWS::SageMaker::Endpoint
- AWS::ServiceCatalog::CloudFormationProduct
- AWS::SES::ConfigurationSet
- AWS::SNS::Topic

- AWS::SQS::Queue
- AWS::StepFunctions::Activity
- AWS::StepFunctions::StateMachine
- AWS::StorageGateway::Gateway
- AWS::Synthetics::Canary
- AWS::Transfer::Server
- AWS::WorkMail::Organization
- AWS::WorkSpaces::Workspace

Properties of an Alarm Status Widget Object

A widget of type alarm can have the following fields within properties.

alarms

An array of alarm ARNs to include in the widget. The array can have 1-100 ARNs.

Type: Array of strings

Required: Yes (when the widget type is alarm).

sortBy

Specifies how to sort the alarms in the widget.

Choose default to sort them in alphabetical order by alarm name.

Choose stateUpdatedTimestamp to sort them first by alarm state, with alarms in ALARM state first, INSUFFICIENT_DATA alarms next, and OK alarms last. Within each group, the alarms are sorted by when they last changed state, with more recent state changes listed first.

Choose timestamp to sort them by the time when the alarms most recently changed state, no matter the current alarm state. The alarm that changed state most recently is listed first.

If you omit this field, the alarms are sorted in alphabetical order.

Type: String

Valid Values: default | stateUpdatedTimestamp | timestamp

Required: No

states

Use this field to filter the list of alarms displayed in the widget to only those alarms currently in the specified states. You can specify one or more alarm states in the value for this field. The alarm states that you can specify are `ALARM`, `INSUFFICIENT_DATA`, and `OK`.

If you omit this field or specify an empty array, all the alarms specified in `alarms` are displayed.

Type: Array of strings

Required: No

title

The title text to be displayed by the widget.

Type: String

Required: No

The following example is an alarm status widget that displays four alarms specified by name, no matter their current state:

```
{
    "type": "alarm",
    "x": 0,
    "y": 0,
    "width": 12,
    "height": 6,
    "properties": {
        "alarms": [
            "arn:aws:cloudwatch:us-east-1:012345678901:alarm:EC2FrontendCPU",
            "arn:aws:cloudwatch:us-east-1:012345678901:alarm:EC2BackendCPU",
            "arn:aws:cloudwatch:eu-west-1:987654321098:alarm:EC2FrontendCPU",
            "arn:aws:cloudwatch:eu-west-1:987654321098:alarm:EC2BackendCPU"
        ],
        "sortBy": "stateUpdatedTimestamp",
        "title": "All EC2 CPU alarms"
    }
}
```

The following example widget specifies the same four alarms, but the widget displays only the alarms that are currently in ALARM or INSUFFICIENT_DATA state:

```
{
    "type": "alarm",
    "x": 0,
    "y": 0,
    "width": 12,
    "height": 6,
    "properties": {
        "alarms": [
            "arn:aws:cloudwatch:us-east-1:012345678901:alarm:EC2FrontendCPU",
            "arn:aws:cloudwatch:us-east-1:012345678901:alarm:EC2BackendCPU",
            "arn:aws:cloudwatch:eu-west-1:987654321098:alarm:EC2FrontendCPU",
            "arn:aws:cloudwatch:eu-west-1:987654321098:alarm:EC2BackendCPU"
        ],
        "sortBy": "stateUpdatedTimestamp",
        "states": [
            "ALARM",
            "INSUFFICIENT_DATA"
        ],
        "title": "EC2 alarms that are not currently OK"
    }
}
```

Properties of a Chart Widget Object

A widget of type `chart` is a multi-data-source chart widget that supports PromQL queries against CloudWatch metrics. Use a chart widget to visualize metrics that are queryable with PromQL, including metrics ingested through OpenTelemetry Line Protocol (OTLP) and AWS vended metrics that have been enriched for OTel. For more information about querying with PromQL, see Query metrics with PromQL.

A widget of type `chart` can have the following fields within `properties`:

view

The visualization to use for the widget.

Valid Values: `line` | `bar` | `pie` | `number` | `solidgauge` | `column` | `legend` | `histogram` | `heatmap` | `scatter`

Type: String

Required: Yes

title

The title to display for the widget.

Type: String

Required: No

description

An optional description for the widget.

Type: String

Required: No

region

The AWS Region to use for all queries in the widget. Each query in the `data.queries` array can override this with its own `region` field.

Type: String

Required: No

data

The queries to plot in the widget. The `data` object contains a `queries` array, where each entry defines one query. For more information about the format, see Chart Widget: Format for Each Query in the Array.

Type: Object

Required: Yes

plotOptions

Visualization options for the widget, such as legend placement, axis configuration, and view-specific style settings (for example, line width or stacked bars). The exact fields available depend on the value of `view`. If you omit `plotOptions`, default settings for the view are used. For the full list of fields, see Chart Widget: plotOptions Reference.

Type: Object

Required: No

Chart Widget: Format for Each Query in the Array

Each entry in the `data.queries` array defines a single query. To create a chart widget that queries CloudWatch metrics, set `type` to `cloudwatch-metrics`, set `language` to either `PromQL` or `SQL`, and provide the query expression as a string in the `query` field. A single chart widget can mix queries written in different languages.

id

An identifier for this query. The ID must be unique within the widget.

Type: String

Required: Yes

type

The kind of query. For PromQL, use `cloudwatch-metrics`.

Valid Values: `cloudwatch-metrics`

Type: String

Required: Yes

language

The query language for this query. Specify `PromQL` to write the query in Prometheus Query Language, or `SQL` to write the query in CloudWatch Metrics Insights SQL.

Valid Values: `PromQL` | `SQL`

Type: String

Required: Yes

query

The query expression to evaluate, as a string. The syntax depends on the `language` field.

For PromQL, provide a Prometheus Query Language expression. For example, `sum by (InstanceId) (rate(CPUUtilization[5m]))`. For information about PromQL syntax and the labels available on AWS vended metrics, see Querying vended AWS metrics with PromQL.

For SQL, provide a CloudWatch Metrics Insights SQL query. For example, `SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId) GROUP BY InstanceId`. For information about Metrics Insights query syntax, see Metrics Insights query components and syntax.

Type: String

Required: Yes

label

The label to display in the legend for this query's series. Specify the special value `__verbose__` to include all metric labels in each series name.

Type: String

Required: No

region

The AWS Region to evaluate this query in. If omitted, the widget's `region` is used.

Type: String

Required: No

accountId

The AWS account to evaluate this query in. Use this for cross-account queries on a monitoring-account dashboard.

Type: String

Required: No

step

The query resolution, in seconds. This controls how frequently the PromQL expression is evaluated across the widget's time range.

Type: Number

Required: No

visible

Set to `false` to hide this query's series from the chart while still keeping the query in the widget definition. The default is `true`.

Type: Boolean

Required: No

Chart Widget: plotOptions Reference

The `plotOptions` object controls how the chart is rendered. It can contain a set of common fields shared by every `view`, plus a `style` object whose shape depends on the chosen `view`. Some views also accept axis (`yAxis`, `xAxis`) and `annotations` objects.

The following table summarizes which fields each `view` supports.

view	style sub-object	yAxis / xAxis	Annotation support
`line`	LineStyle	Yes	Yes (vertical and horizontal)
`bar`	BarStyle	Yes	Yes (vertical and horizontal)
`column`	BarStyle	Yes	No
`pie`	PieStyle	No	No
`number`	NumberStyle	No	No
`solidgauge`	GaugeStyle	No	Yes (horizontal)
`scatter`	ScatterStyle	Yes	Yes (vertical and horizontal)
`histogram`	BarStyle + HistogramStyle	Yes	No

view	style sub-object	yAxis / xAxis	Annotation support
heatmap	HeatmapStyle	Yes	No

In the preceding table, the **style sub-object** column uses shorthand shape names (for example, `LineStyle`, `BarStyle`, and `PieStyle`). You set these styles in the `plotOptions` object using the corresponding fields documented in the following sections (for example, `lineOptions`, `barOptions`, and `pieOptions`).

Common plotOptions fields

The following fields are valid in `plotOptions` for every `view`.

legend

Configures the legend. The object contains the following fields:

- position– Where the legend appears on the widget. Valid values are `bottom` and `right`.
- show– Boolean. Set to `false` to hide the legend.

Type: Object

Required: No

View-specific style fields

The shape of the `style` object depends on the value of `view`.

lineOptions

Used when `view` is `line`. Configures line rendering. The object contains the following fields:

- width– Number. Line width in pixels.
- pattern– Line stroke pattern. Valid values are `solid`, `dashed`, and `dotted`.
- spline– Boolean. If `true`, draws a smoothed curve through the data points.
- filled– Boolean. If `true`, fills the area below the line.
- stacked– Boolean. If `true`, stacks filled areas across series. Use with `filled: true` for a stacked area chart.

- color– A six-digit HTML hex color.

Type: Object

Required: No

barOptions

Used when `view` is `bar`, `column`, or `histogram`. Configures bar rendering. The object contains the following fields:

- stacked– Boolean. If `true`, bars from different series are stacked.
- orientation– Bar direction. Valid values are `vertical` and `horizontal`.

Type: Object

Required: No

pieOptions

Used when `view` is `pie`. The object contains the following fields:

- innerSize– String. A percentage like `"50%"`. Set to `"0%"` for a solid pie, or a non-zero value for a donut chart.

Type: Object

Required: No

numberOptions

Used when `view` is `number`. The object contains the following fields:

- sparkline– Boolean. If `true`, displays a sparkline below the number.
- truncate– Boolean. If `true`, truncates large numbers (for example, displays `1.2K` instead of `1234`).

Type: Object

Required: No

gaugeOptions

Used when `view` is `solidgauge`. The object contains the following fields:

- min– Number. The minimum value of the gauge.

- max– Number. The maximum value of the gauge.

Type: Object

Required: No

colorOptions

Used when `view` is `heatmap`. Configures the color gradient. The object contains the following fields:

- colorPreset– String. Name of a built-in color scale.
- minColor– A six-digit HTML hex color used at the low end of a custom gradient.
- midColor– A six-digit HTML hex color used at the midpoint of a custom gradient.
- maxColor– A six-digit HTML hex color used at the high end of a custom gradient.

Type: Object

Required: No

histogramOptions

Used when `view` is `histogram`, in addition to `barOptions`. The object contains the following fields:

- bucketCount– Number. Number of buckets to divide the data into.
- bucketSize– Number, or the string `"auto"` to size buckets automatically.
- combineSeries– Boolean. Combines all series into a single histogram.

Type: Object

Required: No

aggregationOptions

Used when `view` is `histogram`. Controls how data points are aggregated before bucketing. The object contains the following fields:

- binInterval– String. The bin width as a duration. For example, `"5m"` for five-minute bins.

Type: Object

Required: No

label

Used with line, bar, and scatter styles to label data points directly on the chart. The object contains the following fields:

- position– Where the label appears relative to its data point. Valid values are `inside`, `outside`, `top`, `bottom`, `left`, and `right`.
- show– Boolean. Set to `true` to display labels.

Type: Object

Required: No

markOptions

Used with line and scatter styles to configure markers drawn at data points. The object contains the following fields:

- enabled– Boolean. Set to `true` to draw markers.
- symbolType– String. Marker shape. For example, `circle`, `square`, or `diamond`.
- symbolSize– Number. Marker size in pixels.
- borderWidth– Number. Marker border width in pixels.
- borderColor– A six-digit HTML hex color for the marker border.
- fillColor– A six-digit HTML hex color for the marker fill.
- fillOpacity– Number between 0 and 1.

Type: Object

Required: No

yAxis and xAxis fields

For `view` values that support axes, you can include `yAxis` and `xAxis` objects in `plotOptions`.

yAxis

An array of one or two axis objects. The first entry configures the left axis, and the second entry (if present) configures the right axis. Each axis object contains the following fields:

- min– Number. Lower bound of the axis.

- max– Number. Upper bound of the axis.
- type– Axis scale. Valid values are `linear`, `logarithmic`, `datetime`, and `category`.
- title– String. Axis label displayed alongside the axis.
- showUnits– Boolean. Whether to show unit suffixes (for example, `K`, `M`) on tick labels.

Type: Array of objects

Required: No

xAxis

A single axis object that configures the x-axis. The object contains the following fields:

- min– Number. Lower bound of the axis.
- max– Number. Upper bound of the axis.
- type– Axis scale. Valid values are `linear`, `logarithmic`, `datetime`, and `category`.
- title– String. Axis label displayed alongside the axis.
- label– String. Tick label format.

Type: Object

Required: No

annotations

For `view` values that support annotations, you can add reference lines or shaded bands to the chart. The `annotations` object can contain the following arrays:

vertical

Annotations on the time axis. Each entry is either an annotation object, or an array of two annotation objects to define a band (shaded region between two values). Each annotation object contains the following fields:

- type– The annotation type. Use `static`.
- label– String. The text displayed next to the annotation.
- value– Number, ISO-8601 timestamp string, or category. The location on the time axis where the annotation appears.
- color– A six-digit HTML hex color used for the annotation line and any fill shading.

- fill– How to use fill shading. Valid values are `before` (shading before the annotation), `after` (shading after the annotation), and `between` (used for band annotations to shade between the two values).

Type: Array

Required: No

horizontal

Annotations on the value axis. Each entry is either an annotation object, or an array of two annotation objects to define a band. Each annotation object contains the following fields:

- type– The annotation type. Use `static`.
- label– String. The text displayed next to the annotation.
- value– Number. The value on the value axis where the annotation appears.
- color– A six-digit HTML hex color used for the annotation line and any fill shading.
- fill– How to use fill shading. Valid values are `above` (shading above the annotation), `below` (shading below the annotation), and `between` (used for band annotations to shade between the two values).

Type: Array

Required: No

Example: PromQL line chart

The following example renders a line chart that plots a PromQL query against AWS vended metrics in `us-east-1`.

```
{
    "type": "chart",
    "x": 0,
    "y": 0,
    "width": 12,
    "height": 6,
    "properties": {
        "view": "line",
        "title": "EC2 CPU by instance",
        "region": "us-east-1",
        "data": {
```

```
            "queries": [
                {
                    "id": "a",
                    "type": "cloudwatch-metrics",
                    "language": "PromQL",
                    "query": "sum by (InstanceId) (rate(CPUUtilization[5m]))"
                }
            ]
        },
        "plotOptions": {
            "legend": {
                "position": "bottom",
                "show": true
            }
        }
    }
}
```

Example: Chart that combines a PromQL query and a SQL query

The following example shows a line chart that plots two queries against the same metric, one written in PromQL and one written in CloudWatch Metrics Insights SQL.

```
{
    "type": "chart",
    "x": 0,
    "y": 6,
    "width": 12,
    "height": 6,
    "properties": {
        "view": "line",
        "title": "EC2 CPU utilization (PromQL and SQL)",
        "region": "us-east-1",
        "data": {
            "queries": [
                {
                    "id": "promql_cpu",
                    "type": "cloudwatch-metrics",
                    "language": "PromQL",
                    "query": "sum by (InstanceId) (rate(CPUUtilization[5m]))",
                    "label": "PromQL"
                },
                {
```

```
                    "id": "sql_cpu",
                    "type": "cloudwatch-metrics",
                    "language": "SQL",
                    "query": "SELECT AVG(CPUUtilization) FROM SCHEMA(\"AWS/EC2\",
 InstanceId) GROUP BY InstanceId",
                    "label": "SQL"
                }
            ]
        },
        "plotOptions": {
            "legend": {
                "position": "right",
                "show": true
            }
        }
    }
}
```

Graphing metrics

Use the CloudWatch console to graph metric data generated by other AWS services. This makes it more efficient to see the metric activity on your services. The following procedures describe how to graph metrics in CloudWatch.

Contents

- Graph a metric
- Merge two graphs into one
- Use dynamic labels
- Modify the time range or time zone format for a graph
- Zoom in on a line graph or stacked area graph
- Modify the y-axis for a graph
- Create an alarm from a metric on a graph
- GetMetricWidgetImage: Metric Widget Structure and Syntax

Graph a metric

You can select metrics and create graphs of the metric data using the CloudWatch console.

CloudWatch supports the following statistics on metrics: `Average`, `Minimum`, `Maximum`, `Sum`, and `SampleCount`. For more information, see Statistics.

You can view your data at different levels of detail. For example, you can choose a one-minute view, which can be useful when troubleshooting. Or, choose a less detailed, one-hour view. That can be useful when viewing a broader time range (for example, 3 days) so that you can see trends over time. For more information, see Periods.

If you are using an account that is set up as a monitoring account in CloudWatch cross-account observability, you can graph metrics from the source accounts linked to this monitoring account. For more information, see CloudWatch cross-account observability.

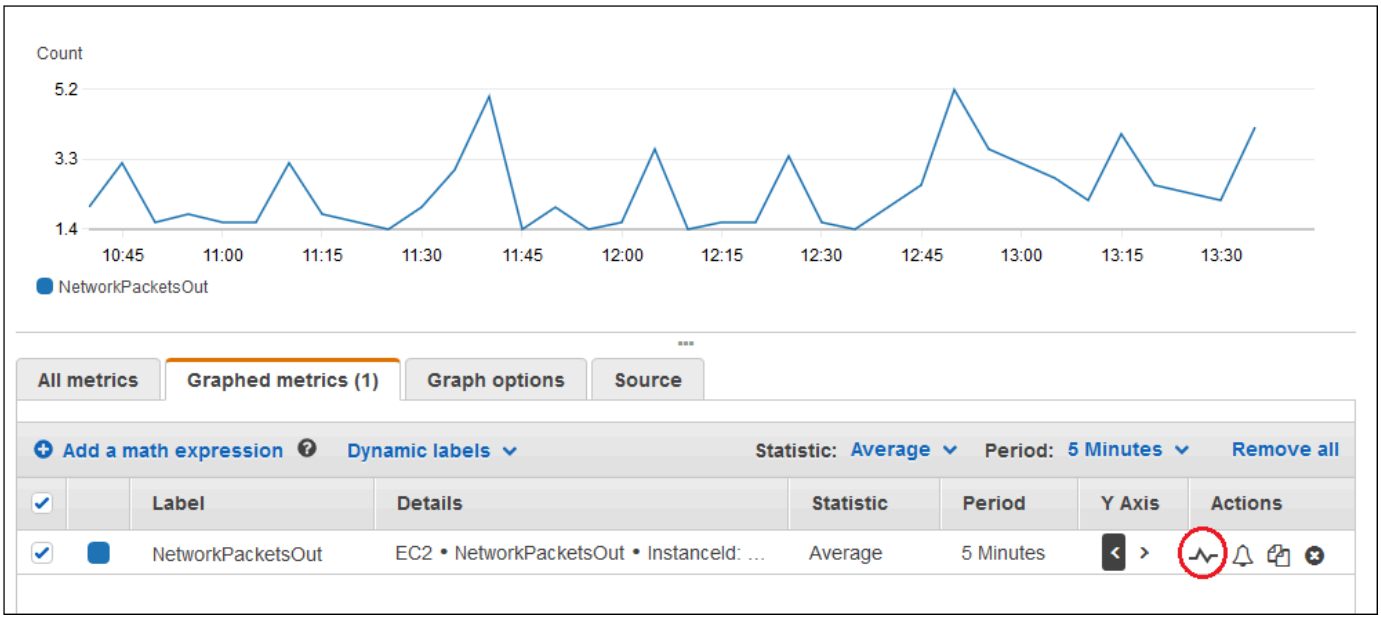
Creating a graph

To graph a metric

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the **Browse** tab, enter a search term in the search field, such as a metric name, account ID, or resource name.

   For example, if you search for the `CPUUtilization` metric, you see the namespaces and dimensions with this metric.
4. Select one of the results for your search to view the metrics.
5. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
6. (Optional) To change the type of graph, choose the **Options** tab. You can then choose between a line graph, stacked area chart, number display, gauge, bar chart, or pie chart.
7. Choose the **Graphed metrics** tab.
8. (Optional) To change the statistic used in the graph, choose the new statistic in the **Statistic** column next to the metric name.

   For more information about CloudWatch statistics, see CloudWatch statistics definitions. For more information about the **p*xx*** percentile statistics, see Percentiles.
9. (Optional) To add an anomaly detection band that shows expected values for the metric, choose the anomaly detection icon under **Actions** next to the metric. For more information about anomaly detection, see Using CloudWatch anomaly detection.



CloudWatch uses up to two weeks of the metric's recent historical data to calculate a model for expected values. It then displays this range of expected values as a band on the graph. CloudWatch adds a new row under the metric to display the anomaly detection band math expression, labeled **ANOMALY_DETECTION_BAND**. If recent historical data exists, you immediately see a preview anomaly detection band, which is an approximation of the anomaly detection band generated by the model. It takes up to 15 minutes for the actual anomaly detection band to appear.

By default, CloudWatch creates the upper and lower bounds of the band of expected values with a default value of 2 for the band threshold. To change this number, change the value at the end of the formula under **Details** for the band.

- (Optional) Choose **Edit model** to change how the anomaly detection model is calculated. You can exclude past and future time periods from being used in the training for calculating the model. It is critical to exclude unusual events system as system outage, deployments, and holidays from the training data. You can also specify the time zone to use for the model for daylight saving time changes.

  For more information, see Editing an anomaly detection model.

For more information about anomaly detection, see Using CloudWatch anomaly detection.

To hide the model from the graph, remove the check mark from the line with the `ANOMALY_DETECTION_BAND` function or choose the `X` icon. To delete the model entirely, choose **Edit model**, **Delete model**.

10. (Optional) As you choose metrics to graph, specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric, and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics**, **Add dynamic label**.

    By default, the dynamic values that you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see Use dynamic labels.

11. To view more information about the metric being graphed, pause the mouse over the legend.

12. Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:

    a. For **Label**, enter a label for the annotation.

    b. For **Value**, enter the metric value where the horizontal annotation appears.

    c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

    d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

    You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

    Repeat these steps to add multiple horizontal annotations to the same graph.

    To hide an annotation, clear the check box in the left column for that annotation.

    To delete an annotation, choose **x** in the **Actions** column.

13. To get a URL for your graph, choose **Actions**, **Share**. Copy the URL to save or share.

14. To add your graph to a dashboard, choose **Actions**, **Add to dashboard**.

Creating a graph of metrics from another data source

You can create a graph that displays resources from data sources other than CloudWatch. For more information about creating connections to these other data sources, see Query metrics from other data sources.

To graph a metric from another data source

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.

   If you haven't already created a connection to the data source that you want, select **Create and manage data sources**, then choose **Create and manage data sources**. For information about the rest of this data source creation process, see Connect to a prebuilt data source with a wizard.
5. The wizard or query editor prompts you for the information necessary for the query. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus and Prometheus data sources, a PromQL query editor box with a query helper appear.
6. When you have finished constructing the query, choose **Graph query**.

   The graph is populated with metrics from the query.
7. (Optional) Horizontal annotations can help graph users more efficiently see when a metric has spiked to a certain level, or whether the metric is within a predefined range. To add a horizontal annotation, choose the **Options** tab and then **Add horizontal annotation**:

   a. For **Label**, enter a label for the annotation.

   b. For **Value**, enter the metric value where the horizontal annotation appears.

   c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

   d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis, if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

8. (Optional) To add this graph to a dashboard, choose **Actions**, **Add to dashboard**.

Updating a graph

To update your graph

1. To change the name of the graph, choose the pencil icon.
2. To change the time range, select one of the predefined values or choose **custom**. For more information, see Modify the time range or time zone format for a graph.
3. To change the statistic, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
4. To change the period, choose the **Graphed metrics** tab. Choose the column heading or an individual value and then choose a different value.
5. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:

   a. For **Label**, enter a label for the annotation.

   b. For **Value**, enter the metric value where the horizontal annotation appears.

   c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

   d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left y-axis or the right y-axis, if the graph includes multiple metrics.

   You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

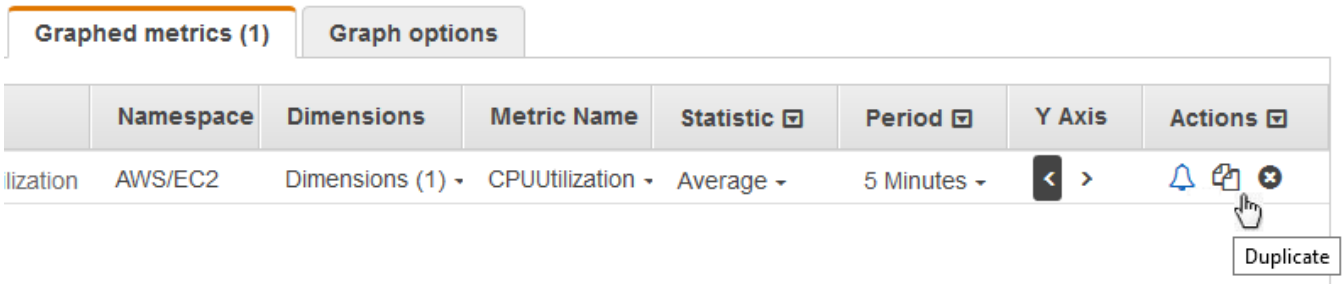
To delete an annotation, choose **x** in the **Actions** column.

6. To change the refresh interval, choose **Refresh options** and then select **Auto refresh** or choose **1 Minute**, **2 Minutes**, **5 Minutes**, or **15 Minutes**.

Duplicating a metric

To duplicate a metric

1. Choose the **Graphed metrics** tab.
2. For **Actions**, choose the **Duplicate** icon.



3. Update the duplicate metric as needed.

Merge two graphs into one

You can merge two different graphs into one, and then the resulting graph shows both metrics. This can be useful if you already have different metrics displayed in different graphs and want to combine them, or you want to easily create a single graph with metrics from different Regions.

To merge a graph into another one, you use either the URL or JSON source of the graph that you want to merge in.

To merge two graphs into one

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Open the graph that you want to merge into another graph. To do so, you can choose **Metrics**, **Classic metrics**, and then choose a metric to graph. Or you can open a dashboard and then

open one of the graphs on the dashboard by selecting the graph and choosing **Open in metrics** from the menu at the upper right of the graph.

3. After you have a graph open, do one of the following:.
   - Copy the URL from the browser bar.
   - Choose the **Source** tab and then choose **Copy**.
4. Open the graph that you want to merge the previous graph into.
5. When you have the second graph open in the **Metrics** view, choose **Actions**, **Merge graph**.
6. Enter the URL or JSON that you previously copied, and choose **Merge**.
7. The merged graphs appear. The y-axis on the left is for the original graph, and the y-axis on the right is for the graph that you merged into it.

   > **Note**
   >
   > If the graph that you merged into uses the **METRICS()** function, the metrics in the graph that was merged in are not included in the **METRICS()** calculation in the merged graph.

8. To save the merged graph to a dashboard, choose **Actions**, **Add to dashboard**.

Use dynamic labels

You can use dynamic labels with your graphs. Dynamic labels add a dynamically updated value to the label for the selected metric. You can add a wide range of values to the labels, as shown in the following tables.

The dynamic value shown in the label is derived from the time range currently shown on the graph. The dynamic part of the label automatically updates when either the dashboard or the graph is refreshed.

If you use a dynamic label with a search expression, the dynamic label applies to every metric returned by the search.

You can use the CloudWatch console to add a dynamic value to a label, edit the label, change the position of the dynamic value within the label column, and make other customizations.

Dynamic labels

Within a dynamic label, you can use the following values relating to properties of the metric:

Dynamic label live value	Description
${AVG}	The average of the values in the time range currently shown in the graph.
${DATAPOINT_COUNT}	The number of data points in the time range that is currently shown in the graph.
${FIRST}	The oldest of the metric values in the time range that is currently shown in the graph.
${FIRST_LAST_RANGE}	The difference between the metric values of the oldest and newest data points that are currently shown in the graph.
${FIRST_LAST_TIME_RANGE}	The absolute time range between the oldest and newest data points that are currently shown in the graph.
${FIRST_TIME}	The timestamp of the oldest data point in the time range that is currently shown in the graph.
${FIRST_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the oldest data point in the time range that is currently shown in the graph.
${LABEL}	The representation of the default label for a metric.
${LAST}	The most recent of the metric values in the time range that is currently shown in the graph.
${LAST_TIME}	The timestamp of the newest data point in the time range that is currently shown in the graph.
${LAST_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the newest data point in the time range that is currently shown in the graph.
${MAX}	The maximum of the values in the time range currently shown in the graph.

Dynamic label live value	Description
${MAX_TIME}	The timestamp of the data point that has the highest metric value, of the data points that are currently shown in the graph.
${MAX_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the data point with the highest value, of those data points that are currently shown in the graph.
${MIN}	The minimum of the values in the time range currently shown in the graph.
${MIN_MAX_RANGE}	The difference in metric values between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph.
${MIN_MAX_TIME_RANGE}	The absolute time range between the data points with the highest and lowest metric values, of those data points that are currently shown in the graph.
${MIN_TIME}	The timestamp of the data point that has the lowest metric value, of the data points that are currently shown in the graph.
${MIN_TIME_RELATIVE}	The absolute time difference between now and the timestamp of the data point with the lowest value, of those data points that are currently shown in the graph.
${PROP('AccountId')}	The AWS account ID of the metric.
${PROP('AccountLabel')}	The label specified for the source account that owns this metric, in CloudWatch cross-account observability.
${PROP('Dim.*dimension_name*')}	The value of the specified dimension. Replace *dimension_name* with the case-sensitive name of your dimension.
${PROP('MetricName')}	The name of the metric.
${PROP('Namespace')}	The namespace of the metric.
${PROP('Period')}	The period of the metric, in seconds.

Dynamic label live value	Description
${PROP('Region')}	The AWS Region where the metric is published.
${PROP('Stat')}	The metric statistic that is being graphed.
${SUM}	The sum of the values in the time range currently shown in the graph.

For example, suppose you have a search expression **`SEARCH(' {AWS/Lambda, FunctionName} Errors ', 'Sum')`**, which finds the `Errors` for each of your Lambda functions. If you set the label to be `[max: ${MAX} Errors for Function Name ${LABEL}]`, the label for each metric is **[max: *number* Errors for Function Name *Name*]**.

You can add as many as six dynamic values to a label. You can use the `${LABEL}` placeholder only once within each label.

Modify the time range or time zone format for a graph

This section describes how you can modify the date, time, and time zone format on a CloudWatch metrics graph. It also describes how you can zoom in on a graph to apply a specific time range. For information about creating a graph, see Graph a metric.

Note

If the time range of a dashboard is shorter than the period used for a graph on the dashboard, the following happens:

- The graph is modified to display the amount of data corresponding one complete period for that widget, even though this is longer than the dashboard time range. This ensures that there is at least one data point on the graph.
- The start time of the period for this data point is adjusted backwards to make sure that at least one data point can be displayed.

Set a relative time range

New interface

To specify a relative time range for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then select the **Relative** tab in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, **Months**.
4. After you specify a time range, choose **Apply**.

Original interface

To specify a relative time range for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Relative** in the upper left corner of the box. You can specify a time range in **Minutes**, **Hours**, **Days**, **Weeks**, or **Months**.

Set an absolute time range

New interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1

hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.

3. Choose **Custom**, and then select the **Absolute** tab in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Original interface

To specify an absolute time range for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose **Absolute** in the upper left corner of the box. Use the calendar picker or text field boxes to specify a time range.
4. After you specify a time range, choose **Apply**.

Set the time zone format

New interface

To specify the time zone for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **Custom** to set your own time range.
3. Choose **Custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local time zone**.
4. After you make your changes, choose **Apply**.

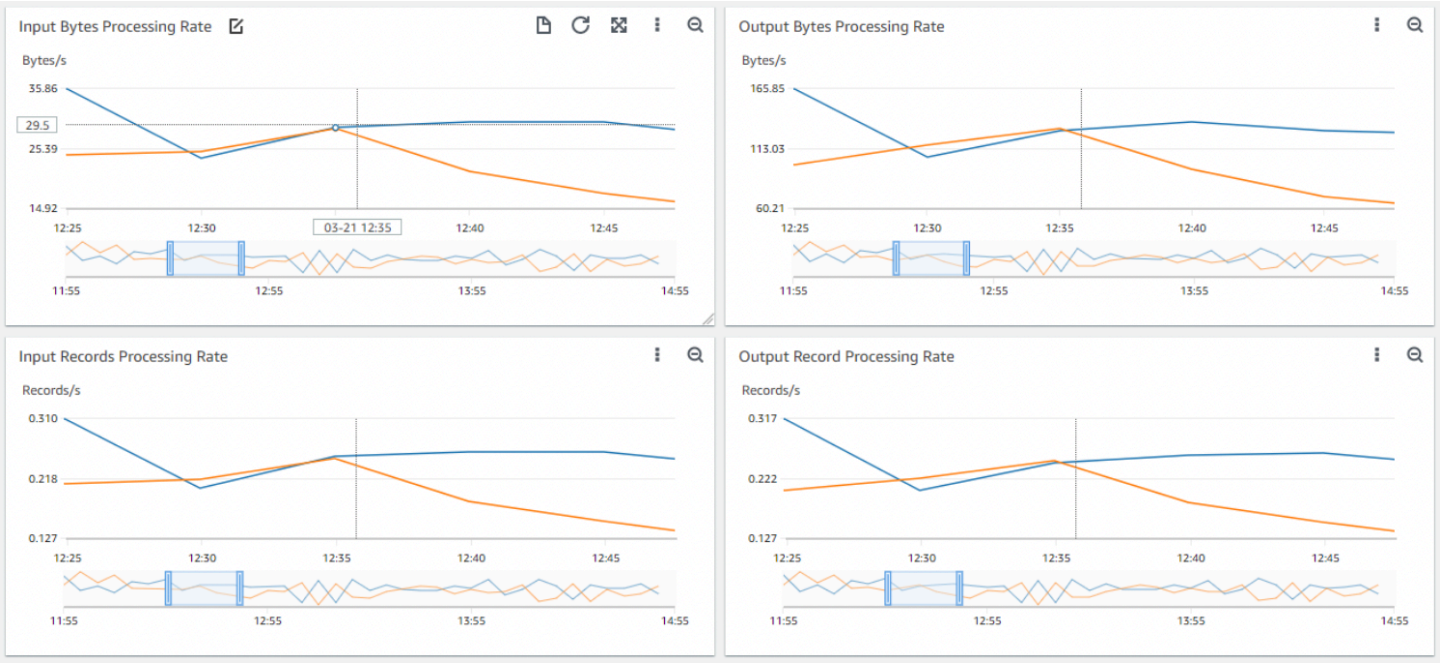
Original interface

To specify the time zone for a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. In the upper right corner of the screen, you can select one of the predefined time ranges, which span from 1 hour to 1 week (**1h**, **3h**, **12h**, **1d**, **3d**, or **1w**). Alternatively, you can choose **custom** to set your own time range.
3. Choose **custom**, and then choose the dropdown in the upper right corner of the box. You can change the time zone to **UTC** or **Local timezone**.

Zoom in on a line graph or stacked area graph

In the CloudWatch console, you can use the mini-map zoom feature to focus on sections of line graphs and stacked area graphs without changing between zoomed-in and zoomed-out views. For example, you can use the mini-map zoom feature to focus on a peak in a line graph, so that you can compare the spike against other metrics in your dashboard from the same timeline. The procedures in this section describe how to use the zoom feature.



In the preceding image, the zoom feature focuses on a spike in a line graph that's related to the input bytes processing rate while also showing other line graphs in the dashboard that focus on sections from the same timeline.

New interface

To zoom in on a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**.
3. Choose **Browse**, and then select a metric or metrics to graph.
4. Choose **Options**, and select **Line** under ***Widget type***.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Original interface

To zoom in on a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**. .
3. Choose **Classic metrics**, and then select a metric to graph.
4. Choose **Graph options**. Under ***Widget type*** select **Line**.
5. Choose and drag on the area of the graph that you want to focus on, and then release the drag.
6. To reset the zoom, choose the **Reset zoom** icon, which looks like a magnifying glass with a minus (-) symbol inside.

Tip

If you already created a dashboard that contains a line graph or stacked area graph, you can go to the dashboard and begin using the zoom feature.

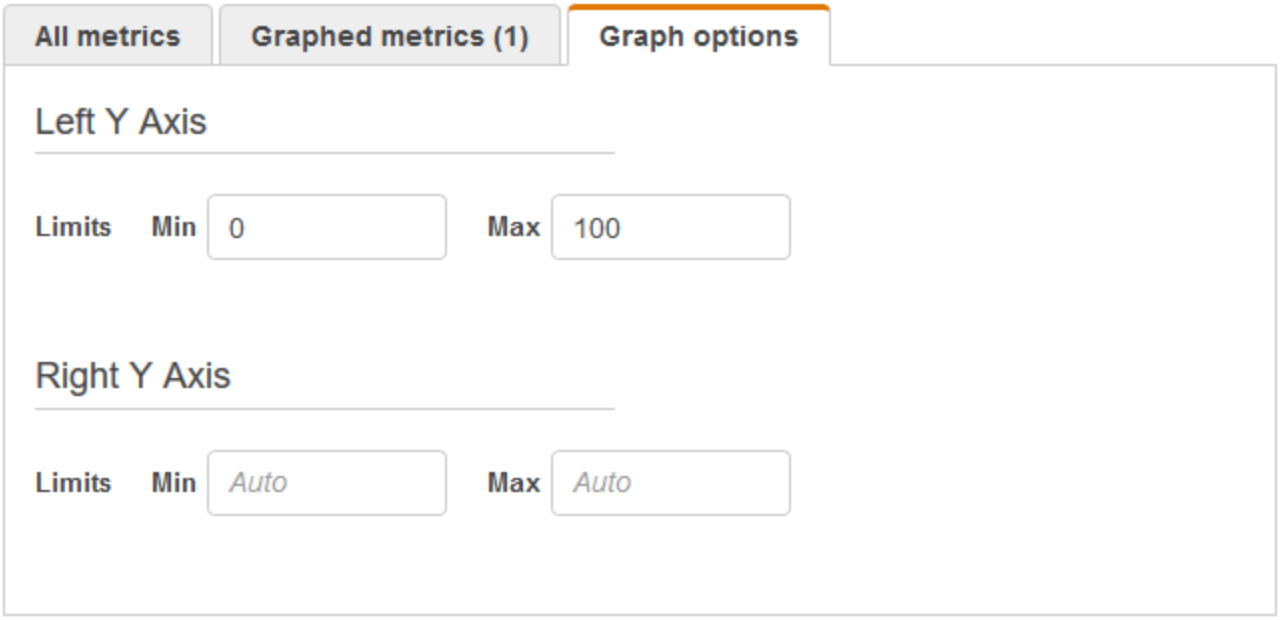
Modify the y-axis for a graph

You can set custom bounds for the y-axis on a graph to help you see the data better. For example, you can change the bounds on a `CPUUtilization` graph to 100 percent so that it's easy to see whether the CPU is low (the plotted line is near the bottom of the graph) or high (the plotted line is near the top of the graph).

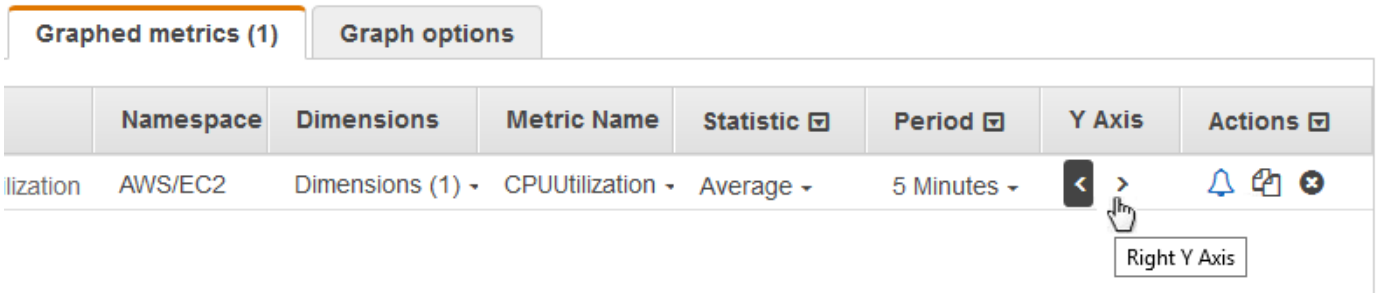
You can switch between two different y-axes for your graph. This is useful if the graph contains metrics that have different units or that differ greatly in their range of values.

To modify the y-axis on a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **Browse** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. On the **Graph options** tab, specify the **Min** and **Max** values for **Left Y Axis**. The value of **Min** can't be greater than the value of **Max**.



6. To create a second y-axis, specify the **Min** and **Max** values for **Right Y Axis**.
7. To switch between the two y-axes, choose the **Graphed metrics** tab. For **Y Axis**, choose **Left Y Axis** or **Right Y Axis**.

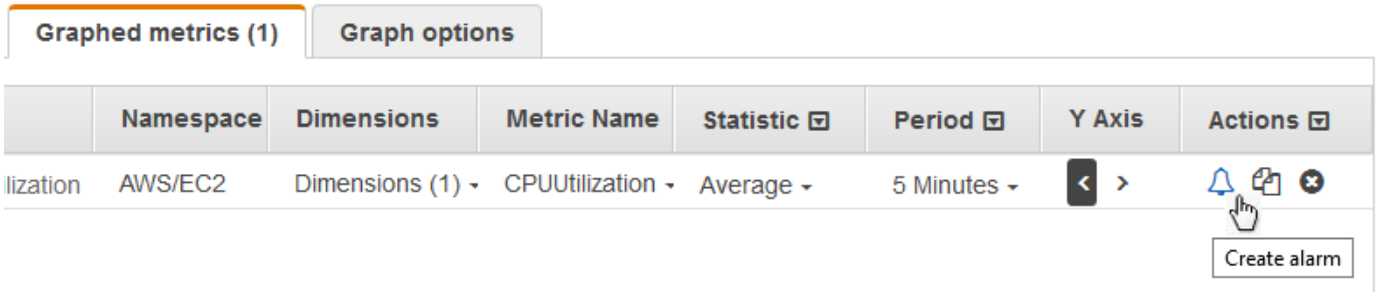


Create an alarm from a metric on a graph

You can graph a metric and then create an alarm from the metric on the graph, which has the benefit of populating many of the alarm fields for you.

To create an alarm from a metric on a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2**) and then a metric dimension (for example, **Per-Instance Metrics**).
4. The **Browse** tab displays all metrics for that dimension in that namespace. To graph a metric, select the check box next to the metric.
5. To create an alarm for the metric, choose the **Graphed metrics** tab. For **Actions**, choose the alarm icon.



6. Under **Conditions**, choose **Static** or **Anomaly detection** to specify whether to use a static threshold or anomaly detection model for the alarm.

   Depending on your choice, enter the rest of the data for the alarm conditions.
7. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here

match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.

8. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
9. Choose **Next**.
10. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.
11. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
12. When finished, choose **Next**.
13. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
14. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

GetMetricWidgetImage: Metric Widget Structure and Syntax

MetricWidget is an input parameter for the https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/API_GetMetricWidgetImage.html API. It is a string in JSON format.

Contents

- Overall Structure
- Format for Each Metric in the Array of Metrics
- Annotation Properties Format
- yAxis Properties Format

Overall Structure

The `MetricWidget` string can include the following parameters:

metrics

The metrics to include in the graph, as a `metrics` array. This can include both raw metric and metric math expressions. One `metrics` array can include 1–100 metrics and expressions. For more information about the format of `metrics`, see Format for Each Metric in the Array of Metrics.

Type: Array of arrays

Required: Yes.

annotations

The horizontal and vertical annotations to add to the graph, as annotations arrays. For more information about the format, see Annotation Properties Format.

Required: No

end

The date and time for the end of the metrics shown in the graph. This can be expressed as either an absolute value, such as **2018-04-25T12:00:00.000Z** or a relative value such as **-PID**.

If you don't specify `end`, the default of `-PT0H` (the current time) is used.

Type: String

Required: No

height

The height of the widget in pixels. The default is 400.

Valid Values: 1–2000

Type: Integer

Required: No, but you should set this if you also set a value for `width`.

legend

Specifies the location and visibility of the graph legend. `legend` contains one field, `position`. The value of `position` can be `bottom`, `right`, or `hidden`. The default is `bottom`.

Type: String

Required: No

liveData

Specify `true` to display *live data* in the widget. Live data is data published within the last minute that has not been fully aggregated. For more information, see Use Live Data.

Type: Boolean

Required: No

period

The default period, in seconds, for all metrics in this widget. This default can be overridden within each metric definition. The default is 300.

Valid Values: 1, 5, 10, 30, 60, and any multiple of 60. 1, 5, 10, and 30 are only for high-resolution metrics.

Type: Integer

Required: No

region

This parameter is optional. If you include it, it must specify the local Region.

Type: String

Required: No

stacked

Specify `true` to display the graph as a stacked line, or `false` to display as separate lines. The default is false.

Type: Boolean

Required: No

start

The date and time for the start of the metrics shown in the graph. This can be expressed as either an absolute value, such as **2018-04-25T12:00:00.000Z** or a relative value such as **-PID**.

If you don't specify `start`, the default of `-PT3H` (three hours ago) is used.

Type: String

Required: No

stat

The default statistic to be displayed for each metric in the array. This default can be overridden within the definition of each individual metric in the `metrics` array.

If you omit this, the default of `Average` is used.

Valid Values: `SampleCount` | `Average` | `Sum` | `Minimum` | `Maximum` | `p??` | `TM(??:??)`, `TC(??:??)` | `TS(??:??)` | `WM(??:??)` | `PR(??:??)` | `IQM`

Type: String that is a valid CloudWatch statistic.

Required: No

theme

The color palette used to style the graph. The default is `light`.

Valid Values: `light | dark`

Type: String

Required: No

timezone

The time zone to use for displaying the times in the graph. The format is + or - followed by four digits. The first two digits indicate the number of hours ahead or behind of UTC, and the final two digits are the number of minutes. For example, `+0130` indicates a time 1 hour and 30 minutes ahead of UTC. The default is `+0000`.

Type: String

Required: No

title

The title to be displayed for the graph.

Type: String

Required: No

view

The display format. Specify `timeSeries` to display this metric as a line graph. Specify `bar` to display the metric as a bar graph. Specify `pie` to display the metric as a pie graph. The default is `timeSeries`.

Valid Values: `timeSeries | bar | pie`

Type: String

Required: No

width

The width of the widget in pixels. The default is 600.

Valid Values: 1–2000

Type: Integer

Required: No, but you should set this if you also set a value for `height`.

yAxis

Limits for the minimums and maximums of the y-axis. This applies to every metric being graphed, unless specific metrics override it. For more information about the format, see yAxis Properties Format.

Type: YAxis object

Required: No

Format for Each Metric in the Array of Metrics

Each item in the `metrics` array is a CloudWatch metric to display in the graph, or to use as part of a math expression displayed in the graph. For more information about math expressions, see Use Metric Math in the Amazon CloudWatch User Guide.

Each metric in the array has the following format:

```
[Namespace, MetricName, Dimension1Name, Dimension1Value, Dimension2Name, Dimension2Value...
            {Options Object}]
```

Namespace

The AWS namespace containing the metric. To use the same namespace as the previous metric in the array, you may specify `"."` for each entry after the first.

Type: String

Required: Yes

MetricName

The name of the CloudWatch metric. To use the same name as the previous metric in the array, you may specify `"."` for each entry after the first.

Type: String

Required: Yes

DimensionName

The name of a dimension to further refine what data is shown. To use the same dimension name as the previous metric in the array, you may specify `"."` for each entry after the first. You may specify zero dimensions for a metric, or up to as many dimensions as the metric supports.

Type: String

Required: No

DimensionValue

The value to use for that dimension for the metric. Required if there is a corresponding dimension name.

Type: String

Required: No, unless there is a corresponding dimension name.

Options Object

Specifies either custom rendering properties to be used for the specified CloudWatch metric, or a math expression to display on the graph. For more information about the format, see Options Object Format.

Type: Options Object

Required: No

Examples

```
// The simplest example, a metric with no dimensions
        [ "AWS/EC2", "CPUUtilization" ]

 // A metric with a single dimension
        [ "AWS/EC2", "CPUUtilization", "InstanceId", "i-01234567890123456" ]

 // A metric with a single dimension and rendering properties
        [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-01234567890123456", { yAxis:
 "right"} ]

 // The following example graphs the DiskReadBytes metric for three instances.
        [ "AWS/EC2", "DiskReadBytes", "InstanceId", "i-01234567890123456" ],
        [ ".", ".", ".", "i-abc" ],
        [ ".", ".", ".", "i-123" ]
```

Options Object Format

Specifies either custom rendering properties to be used for the specified CloudWatch metric, or a math expression to display on the graph.

If this object is specified as part of a CloudWatch metric in the `metrics` array, it sets custom rendering properties for this metric and overrides the defaults used for the whole graph.

You can also specify this object to add a math expression to the graph. In this case, the other settings in this object specify the display options for the result of the math expression.

This section describes the format of these options objects.

color

The six-digit HTML hex color code to be used for this metric or expression.

Type: String

Required: No

expression

A math expression to display. For more information about supported math expression functions and format, see Metric Math Syntax and Functions in the Amazon CloudWatch User Guide.

Type: String that is a valid CloudWatch metric math expression.

Required: Yes if this is an expression.

label

The label to display for this metric or expression in the graph legend. If this is not specified, the metric is given an auto-generated label that distinguishes it from the other metrics in the widget.

Type: String

Required: No

id

An identifier for this metric or expression, which must be unique within this widget. The id can be used as a variable to represent this metric or expression within math expressions. Valid characters are letters, numbers, and underscore. The first character must be a lowercase letter.

Type: String

Required: No

period

The period for this metric, in seconds. If specified, this overrides the default period used for other metrics in this graph. This parameter is not applicable for math expressions.

Valid Values: 1, 5, 10, 30, 60, and any multiple of 60. 1, 5, 10, and 30 are only for high-resolution metrics.

Type: Integer

Required: No

stat

The statistic to be displayed for this metric, if it is to be different than the statistic used for the other metrics in the graph. This parameter is not applicable for math expressions.

Valid Values: `SampleCount` | `Average` | `Sum` | `Minimum` | `Maximum` | p*??*

Type: String that is a valid CloudWatch statistic.

Required: No

visible

Specifies whether this metric or expression is shown on the graph. The default is `true`.

Setting `visible` to `false` is useful if you want to hide the raw metrics that are used in math expressions, and show only the expression results on the graph.

Type: Boolean

Required: No

yAxis

Where on the graph to display the y-axis for this metric or expression. The default is `left`.

Valid Values: `left` | `right`

Type: String

Required: No

Example

In the following example, CloudWatch retrieves a custom `apiLatency` metric. At the top, the p50 statistic is specified to show the median value. Next, for the same metric on the same instance (this

is specified by the four fields that are just periods), the Average value is graphed. Next is an options object with a math expression, showing the halfway value of the two metrics. Finally, another expression shows the rate of change.

To show only the results of the two expressions on the graph and hide the raw metrics, you could change the first two instances of `visible` to `false`.

```
{
        "metrics": [
            [
                "MyNamespace",
                "apiLatency",
                "InstanceId",
                "i-0987654321abcdef0",
                {
                    "id": "m1",
                    "stat": "p50",
                    "label": "Median value",
                    "visible": true,
                    "color": "#dddddd",
                    "yAxis": "left",
                    "period": 300
                }
            ],
            [
                ".",
                ".",
                ".",
                ".",
                {
                    "id": "m2",
                    "stat": "Average",
                    "label": "Average value",
                    "visible": true,
                    "color": "#cccccc",
                    "yAxis": "left",
                    "period": 300
                }
            ],
            [
                {
                    "expression": "(m1+m2)/2",
                    "id": "e1",
```

```
                    "label": "Half way between average and median",
                    "visible": true,
                    "color": "#000000",
                    "yAxis": "left"
                }
            ],
            [
                {
                    "expression": "RATE(e1)",
                    "yAxis": "right",
                    "label": "rate of change of the half way point"
                }
            ]
        ]
    }
```

Annotation Properties Format

A single graph can have multiple horizontal and vertical annotations. All horizontal annotations are specified in one `horizontal` field, and all vertical annotations are specified in one `vertical` field.

horizontal

An array of horizontal annotations. Horizontal annotations have several options for fill shading, including shading above the annotation line, shading below the annotation line, and "band" shading that appears between two linked annotation lines as part of a single annotation. Each horizontal annotation in the array that does not have band shading has the following format:

```
{value, label, color, fill, yAxis, visible}
```

Each horizontal annotation that does have band shading has the following format:

```
[{value, label, color, yAxis, visible}, {value, label}]
```

vertical

An array of vertical annotations. Vertical annotations have several options for fill shading, including shading before the annotation line, shading after the annotation line, and "band" shading that appears between two linked annotation lines as part of a single band annotation. Each vertical annotation in the array that does not have band shading has the following format:

```
{value, label, color, fill, visible}
```

Each vertical annotation that does have band shading has the following format:

```
[{value, label, color, visible}, {value, label}]
```

The `horizontal` array can include the following fields.

value

The metric value in the graph where the horizontal annotation line is to appear. On a band shading annotation, the two values for `Value` define the upper and lower edges of the band.

On a graph with horizontal annotations, the graph is scaled so that all visible horizontal annotations appear on the graph.

Type: Float

Required: Yes, if horizontal annotations are used.

label

A string that appears on the graph next to the annotation.

Type: String

Required: No

color

The six-digit HTML hex color code to be used for the annotation. This color is used for both the annotation line and the fill shading.

Type: String

Required: No

fill

How to use fill shading with the annotation. Valid values are `above` for shading above the annotation, `below` for shading below the annotation, and `none` for no shading. If `fill` is omitted, there is no shading.

The exception is an annotation with band shading. These annotations always have shading between the two values, and any value for `fill` is ignored.

Type: String

Required: No

visible

Set this to `true` to have the annotation appear in the graph, or `false` to have it be hidden. The default is `true`.

Type: Boolean

Required: No

yAxis

If the graph includes multiple metrics, specifies whether the numbers in `Value` refer to the metric associated with the left Y-axis or the right Y-axis. Valid values are `right` and `left`.

Type: String

Required: No

The `vertical` array can include the following fields.

value

The time stamp where the vertical annotation line is to appear. This must be specified as an absolute time stamp, such as `2018-08-28T15:25:26Z`. On a band shading annotation, the two values for `Value` define the beginning and ending edges of the band.

Type: String

Required: Yes, if vertical annotations are used.

label

A descriptive string that appears on the graph next to the annotation.

Type: String

Required: No

color

The six-digit HTML hex color code to be used for the annotation. This color is used for both the annotation line and the fill shading.

Type: String

Required: No

fill

How to use fill shading with the annotation. Valid values are `before` for shading before the annotation, `after` for shading after the annotation, and `none` for no shading. If `fill` is omitted, there is no shading.

The exception is an annotation with band shading. These annotations always have shading between the two values, and any value for `fill` is ignored.

Type: String

Required: No

visible

Set this to `true` to have the annotation appear in the graph, or `false` to have it be hidden. The default is `true`.

Type: Boolean

Required: No

Examples

```
// A single horizontal annotation with fill shading above the annotation line, based on
 the metric associated with the right Y-axis

"annotations": {
     "horizontal": [
         {
              "visible":true,
              "color":"#9467bd",
              "label":"Critical range",
              "value":20,
              "fill":"above",
```

```
             "yAxis":"right"
         }
    ]
}

// A horizontal band annotation. Each value has a label, but other parameters for the
 band need to be specified only with the first number

"annotations": {
    "horizontal": [
        [
             {
                 "label":"Band top",
                 "value":200,
                 "color":"#9467bd",
                 "visible":true,
                 "yAxis":"right"
             },
             {
                 "value":95.5,
                 "label":"Band bottom"
             }
         ]
     ]
}

// A single vertical annotation with fill shading after the annotation line

"annotations": {
    "vertical": [
        {
            "visible": true,
            "color": "#9467bd",
            "label": "Bug fix deployed",
            "value": "2018-08-28T15:25:26Z",
            "fill": "after"
        }
    ]
}


// A vertical band annotation. Each annotation line has a label, but other parameters
 for the band are specified only with the first value
```

```
"annotations": {
    "vertical": [
        [
            {
                "label": "Band start",
                "value": "2018-08-27T15:25:26Z",
                "color": "#9467bd",
                "visible": true
            },
            {
                "value": "2018-08-28T15:25:26Z",
                "label": "Band end"
            }
        ]
    ]
}
```

yAxis Properties Format

Defines the minimum and maximum values for the Y-axis of the graph. Set this within the `MetricWidget` object to affect all metrics in the widget. To override the widget settings for a particular metric, set it in the options object for that metric in the `metrics` array.

left

Optional `min`, `max`, `label`, and `showUnits` settings for the left Y-axis.

Type: YAxis object

Required: No

right

Optional `min`, `max`, `label`, and `showUnits` settings for the right Y-axis.

Type: YAxis object

Required: No

Each of the `left` and `right` objects can include the following parameters:

min

The minimum value for this Y-axis.

Type: Float

Required: No

max

The maximum value for this Y-axis.

Type: Float

Required: No

label

The label for this Y-axis.

Type: String

Required: No

showUnits

Specifies whether to display the unit of the metrics on this Y-axis. The default is `true`.

Type: Boolean

Required: No

Example

```
{
  left: {
    min: 0,
    max: 100,
    label: "Duration (ms)",
    showUnits: false
  },
```

```
  right: {
    min: 0,
    label: "Invocations"
  }
}
```

Explore related telemetry

Computer systems can generate a large amount of telemetry, including both metrics and logs, and complex systems even more so. When looking at a specific set of telemetry, it can be a challenge to find other telemetry related to your initial set. It can take advanced training to gain the skills needed to find issues and troubleshoot them. Because systems are complex, understanding what is going on can involve viewing metrics and logs from many different services and resources, requiring context switching and navigating between systems.

The Amazon CloudWatch **Explore related** feature offers access to AWS resource relationships, related metrics, and logs across service consoles, enhancing observability and efficiency for operators of all skill levels. When viewing alarms or anomalies in CloudWatch dashboards, or metrics in AWS, users can quickly find and view metrics and logs for related resources in your system.

CloudWatch provides visibility into metrics and logs tied to specific resources. The **Explore related** pane extends that by allowing you to correlate your infrastructure resources to your workloads with all of their associated telemetry. This gives you quick access to the information you need to troubleshoot infrastructure-related issues. You view the relationship between resources, and their related telemetry in the **Explore related** pane. The **Explore related** pane is accessed from CloudWatch or from other AWS consoles showing resources or telemetry.

> **Note**
>
> **Explore related** is currently limited in accounts set up as monitoring accounts in CloudWatch cross-account observability. You should access **Explore related** from the source accounts where the resources are originally created and managed. In source accounts, you can navigate between connected resources, and view related logs and metrics.

The following topics discuss the details of exploring related telemetry.

Topics

- What is related telemetry?
- How to access the Explore related pane
- Navigating related telemetry
- Using the topology map
- Finding a specific resource
- Permissions and prerequisites needed to view and explore related telemetry
- How does CloudWatch find related telemetry?
- AWS services that support related telemetry
- How to add related information to custom telemetry sent to CloudWatch

What is related telemetry?

Related telemetry is metrics and log data from resources that are related to the current resource or service. Traditionally, you might look at the metrics and logs that are related to a single load balancer, or all telemetry related to Amazon EC2. The **Explore related** feature in Amazon CloudWatch adds an interactive *topology map*. The map is a resource-centric view where you can find metrics and logs for a specific resource, but you can also see how that resource is connected to other resources.

For example, if you are looking at the telemetry for a load balancer in the **Explore related** pane, besides the metrics and logs associated with that load balancer, the maps shows you the target groups for that load balancer. Selecting one of the target groups will then show you the Amazon EC2 instances associated with that target group. At each step in this process, the telemetry, including metrics and logs, for the selected resources are shown, making it easy to quickly find the telemetry you are looking for, or to explore the telemetry, looking for the cause of a specific issue.

How to access the Explore related pane

Within the CloudWatch console, there are multiple ways to access telemetry related to your current view. For example, if you are looking at a graph on a dashboard, and you want to view telemetry related to that graph or an aspect of the graph, you can choose to explore related data directly from that graph. From many places in the console, you can choose an **Explore related** menu item, or select a compass icon () to show the **Explore related** pane.

You can access the exploring experience from entry points throughout the CloudWatch console (and other AWS consoles), including:

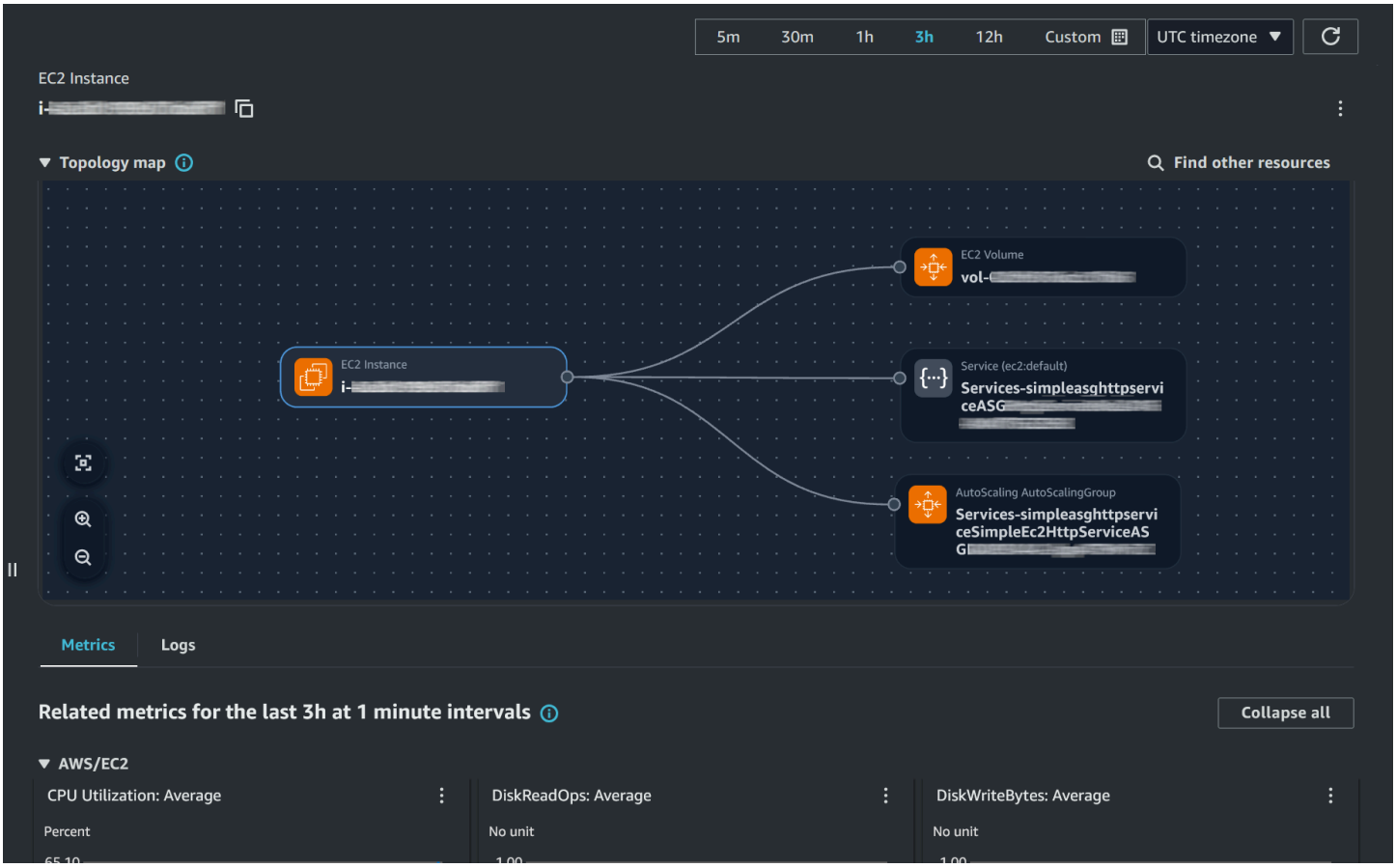
- **Metrics navigation** – When you choose **Metrics** then **Classic metrics** from the CloudWatch console left hand menu, the tile for any supported service or metric source will show the compass icon that brings up related telemetry in the bottom right corner.
- **Metric legend** – When viewing any metric graph (in CloudWatch or other AWS consoles), hovering over the legend of the graph shows information about the data, as well as an **Explore related** button that brings up the related telemetry.
- **Metric datapoints** – When viewing any metric graph, hovering over a datapoint in the graph shows information about the metric, as well as a compass icon, to bring up the related telemetry.
- **Metric search** – When searching for metrics in CloudWatch, if you choose a found metric name, you can select **Explore related** from the menu that pops up, which will bring up the related telemetry.
- **Console toolbar** – In many AWS console pages, the console toolbar (usually in the upper right of the console) includes a CloudWatch service icon, which will bring up CloudWatch tools, including the **Explore related** pane.

Depending on where you access the pane from, the default context of the pane will show an appropriate filter, if possible.

Navigating related telemetry

When you select any of the entry points to the **Explore related** pane, it appears on the right side of the CloudWatch console. This pane gives you access to view and find telemetry related to *entities*, in your system. An entity is a resource, such as an Amazon EC2 instance, or a service, such as an application that you have built. You can work within this pane without interrupting your current workflow, because it opens to the side of your initial page.

The following image shows the **Explore related** pane focused on a single Amazon EC2 instance, and the associated entities.



The top of the **Explore related** pane is a visual *topology map* (map) of the current entity and other related entities. The currently selected entity sets the focus for the pane. There are two ways to select an entity.

- Topology map – The *map* is a visual display of the current entity with focus. It also displays any related entities, allowing you to navigate around the set of resources and services that are related to each other.
- Find other resources – You can use the **Find other resources** button to filter and search for entities to use as the focus.

The bottom part of the pane shows you an automated search of metrics and logs for the current focal entity.

By default, the focus is set to an entity matching the location from which you accessed the **Explore related** pane. For example, if you accessed it by clicking the compass icon associated with a metric from a specific Amazon EC2 instance, then the focus will be set to that Amazon EC2 instance.

When you select an AWS resource for the focus in the **Explore related** pane, you can navigate to the resource-specific console for the selected resource. For example, if you have selected an Amazon EC2 instance, you can select the **View in EC2 console** link to open the Amazon EC2 console with that instance selected.

When you set a focus, the metrics and logs are automatically filtered to show the telemetry related to your focus.

- **Metrics** – Each metric is shown as a graph for the time period that you have chosen.

  Much like any dashboard graph in CloudWatch, you can hover over or select a graph to get more information about the metric graph, and to see options including viewing in CloudWatch metrics. Choosing to view in CloudWatch will open the metrics view with the same viewing context as the **Explore related** pane, including resource and time range.

- **Log patterns** – CloudWatch analyzes the log groups associated with the focal resource and shows common patterns in those logs. For more information about log patterns, see Pattern analysis in the *Amazon CloudWatch Logs User Guide*.

  You can select **Compare timerange** to choose another time range and compare logs across the two time ranges.

  You can select **View in Logs Insights** to analyze the logs in CloudWatch Logs Insights with the same options as your current view, including resource, log group, and time range. For more information, see Analyzing log data with CloudWatch Logs Insights in the *Amazon CloudWatch Logs User Guide*.

- **Log groups** – The log groups that contain the logs are shown.

  You can select the log groups and then perform one of the following actions:

  - Choose **Start tailing in Live Tail** to view a streaming list of new log events as they are ingested for the selected log groups. The Live Tail session is started in the CloudWatch console. For more information on Live Tail, see Troubleshoot with CloudWatch Logs LiveTail in the *Amazon CloudWatch Logs User Guide*.
  - Choose **Query in Logs Insights** to open Logs Insights with a query scoped to just those log groups, applying your current context, including resources and time range.

Using the topology map

The *topology map* (map) is a visual display of the current focal entity and its related resources or services. You can use this interactive visualization to see the connections between different resources and services, and explore the relationships between components in your system. For example, if you are viewing *load balancer* resource, the map will show connected *target group* resources. Selecting a target group will display the associated instances. The visualization of connectivity helps operators understand and explore the relationships between different resources and services in your system.

You can drag the map, and zoom in and out, to see more of the associated entities, or to focus on fewer entities.

When you select an associated entity, like a target group, the pane's focus shifts to show telemetry for that entity. The map updates to center on the selected target group, displaying its connections to other entities, such as the load balancer and and any Amazon EC2 instances that are specified in that target group. As you navigate through different entities in the map, the metrics and logs at the bottom of the pane dynamically update, providing you with relevant telemetry for the newly selected resources.

Finding a specific resource

If a resource doesn't appear on the topology map, you can use the **Find other resources** feature to locate it. You can filter resources by tag or type, then select the ones you are looking for. After you have found resources to focus on, you are returned to the topology map, with those resources selected, to browse associated entities and telemetry.

> **Note**
>
> There are many reasons why you might not see your resources on the topology map. For example:
>
> - It's not related to the current focal entity.
> - You don't have permissions to access associated entities or telemetry.
> - The resource or service may not support telemetry or associated entities.

By using **Find other resources**, you can discover and visualize resources that may not be directly connected or visible in the current map. This ensures that you can access and analyze all relevant components of your infrastructure.

Select a resource with Find resources.

1. Open the **Explore related** pane from any of the entry points in the CloudWatch console.
2. Choose **Find resources**.
3. Choose the time frame for which you want to view logs or metrics.
4. Choose **Resource types**, then select the type of the resource you want to focus on from the drop down list, for example, **EC2 instances**.
5. Optionally, filter the set of resources by providing a tag to filter on. You can do this by selecting the **Filter resources by tags** filter, or by choosing the label that says **5 tags found** (the number will depend on the tags in your system). This gives you a list of tags to choose from.

   After you have selected the tags, the list of resources is automatically filtered to only those that are associated with those tags.
6. Optionally, choose one or more specific resources from the ones found that match your filters.
7. Choose **Show on map** to return to the **Topology map** with your resources selected.

Your list of **Metrics** and **Logs** is now filtered to just logs and metrics that are associated with that resource type. You can choose the **Metrics** or **Logs** tab to view the kind of telemetry that you want to view.

Permissions and prerequisites needed to view and explore related telemetry

To explore related telemetry, you must be getting entity information with the telemetry from your workloads, and you must have the proper permissions to view that data.

Many services send entity information automatically. For workloads that use the CloudWatch agent, you must have at least version `1.300049.1` of the agent, and you must configure it correctly. For information about configuring the agent, see How to add related information to custom telemetry sent to CloudWatch. For workloads running on Amazon EKS, you must have at least version `v2.3.1-eksbuild.1` of the Amazon CloudWatch Observability EKS add-on. For more information about this add-on, see Quick start: OTel Container Insights on Amazon EKS.

To explore related telemetry you must be signed in with certain permissions. Exploring related telemetry is a read-only activity, and requires at least read-only access to CloudWatch.

The permissions needed for viewing associations between telemetry and entities are: `logs:ListLogGroupsForEntity`, `logs:ListEntitiesForLogGroup`, `cloudwatch:ListEntitiesForMetric`, and `application-signals:ListObservedEntities`.

Each of the following AWS managed policies will provide the CloudWatch permissions needed to access related telemetry in the CloudWatch console:

- CloudWatchFullAccessV2 – Provides full access to CloudWatch.
- CloudWatchReadOnlyAccess – Provides read-only access to CloudWatch.
- ReadOnlyAccess – Provides read-only access to AWS services and resources.

Additionally, you must have at least read-only access (`Describe*` and `Get*`) to any resources in the topology map, in order for CloudWatch to discover and display relationships.

For more details about using policies to control access, see Identity-based policy examples for Amazon CloudWatch.

How does CloudWatch find related telemetry?

The CloudWatch **Explore related** pane shows you metrics and logs that are related to each other, but how does that work?

Metrics and logs that are sent to CloudWatch can include an optional *entity* to which they are related. Typically, the entity will be a representation of what the telemetry is about. For example, a metric about CPU usage is about an Amazon EC2 instance, and will use that instance as its entity. When you view that metric in the **Explore related** pane, it shows you other telemetry for that same instance.

The *topology map* (map) in the **Explore related** pane displays the currently selected resource, along with *related resources*. For AWS resources, CloudWatch automatically displays other resources that it knows are related. For example, if you are viewing an Amazon EC2 instance, the map will also display any Amazon EBS volumes that are attached to the instance. Selecting a volume shows telemetry for the volume, and the map is updated to display resources related to the volume. It also displays resources that are part of the same *service*.

The entity information associated with your telemetry defines the resource that the telemetry is associated with, such as the Amazon EC2 instance. However, it can also include contextual data about the resource. For example, if you have a website application that includes resources such as an Amazon EC2 instance and a database, the entity information can include the website application as a service. In this case, the topology map shows the service as a related entity, and when you select it, it displays the instances and database. This can make finding all the telemetry for a service simpler.

> **Note**
>
> CloudWatch must have received telemetry with entity information within the last three hours in order to find related resources and telemetry.

Where does the entity data come from?

There are different ways that CloudWatch gets entities for telemetry:

- Most telemetry sent to CloudWatch from AWS services are associated with resources automatically. For a complete list of supported resources, see AWS services that support related telemetry.
- The CloudWatch agent automatically adds entity information to the telemetry that it sends to CloudWatch.

  > **Note**
  >
  > You may need to update your CloudWatch agent to the latest version to include entity data. For more information, see Collect metrics, logs, and traces using the CloudWatch agent, and Configure CloudWatch agent service and environment names for related entities.

- When you are submitting your own telemetry, you can add entity information to the data. For more information, see How to add related information to custom telemetry sent to CloudWatch.
- CloudWatch makes a best effort to recognize the entity information associated with other telemetry (for example, custom telemetry that you send to CloudWatch without any entity information).

Where does service data come from?

Besides recognizing the natural connections between resources, such as an instance resource and an attached volume resource, CloudWatch can also group resources by *service*. For example, a service might be a website application. An Amazon EC2 instance with a web server, and another with a database might both be part of the same service, and are connected on the topology map based on that service.

There are different ways that CloudWatch gets a service name for telemetry, including:

- Application signals or otel instrumented telemetry use the `OTEL_SERVICE_NAME` environment variable used by supported OpenTelemetry instrumentation libraries to set the service name.
- The CloudWatch agent configuration allows configuring a service name. For more information, see Configure CloudWatch agent service and environment names for related entities.
- Kubernetes workloads use a corresponding name from the cluster, such as the Deployment, ReplicaSet, Pod, or Container, for the service name.
- For Amazon EC2 workloads, the service can come from tags (the `service`, `application`, or `app` tags).

> **Note**
>
> To use tags to generate service names, you must first set up instance metadata for the Amazon EC2 instance.

- When you are submitting your own telemetry, you can add service information to the data. For more information, see How to add related information to custom telemetry sent to CloudWatch.
- When it cannot use the above, CloudWatch uses the name of the IAM role that sends the metrics as the service name. This, for example, can provide a service name for Amazon ECS telemetry.

AWS services that support related telemetry

The following table lists the AWS services that support related entity information in their CloudWatch telemetry. Services or resources that are not listed in the table do not have related entity information exposed in CloudWatch.

Note

For services that use the CloudWatch agent, you may need to update the agent to the latest version to get related telemetry. For information about sending related entity information with your own custom metrics, see How to add related information to custom telemetry sent to CloudWatch.

AWS Service	Resource	Metrics	Logs
Amazon API Gateway	`AWS::ApiGateway::Method`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::Resource`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::RestApi`	Yes	No
Amazon API Gateway	`AWS::ApiGateway::Stage`	Yes	Yes
Amazon API Gateway	`AWS::ApiGateway::VpcLink`	Yes	No
Amazon API Gateway V2	`AWS::ApiGatewayV2::Integration`	Yes	No
Amazon API Gateway V2	`AWS::ApiGatewayV2::Route`	Yes	No

AWS Service	Resource	Metrics	Logs
Amazon API Gateway V2	`AWS::ApiGatewayV2::Stage`	No	Yes
Amazon API Gateway V2	`AWS::ApiGatewayV2::Api`	Yes	No
AWS App Runner	`AWS::AppRunner::Service`	No	Yes
AWS Transform MGN	`AWS::MGN::SourceServer`	Yes	No
Amazon WorkSpaces Applications	`AWS::AppStream::Fleet`	Yes	Yes
AWS AppSync	`AWS::AppSync::GraphQLApi`	Yes	Yes
AWS B2B Data Interchange	`AWS::B2BI::Transformer`	No	Yes
AWS Backup gateway	`AWS::BackupGateway::Hypervisor`	No	Yes
Amazon Bedrock	`AWS::Bedrock::KnowledgeBase`	No	Yes
Amazon Bedrock	`AWS::Bedrock::ModelId`	Yes	No
Amazon Chime	`AWS::Chime::SipMediaApplication`	No	Yes

AWS Service	Resource	Metrics	Logs
Amazon Chime	AWS::Chime::VoiceConnector	No	Yes
AWS Clean Rooms	AWS::CleanRooms::Membership	No	Yes
AWS CloudFormation		No	Yes
AWS CloudFormation Hooks		No	Yes
Amazon CloudFront	AWS::CloudFront::Distribution	Yes	Yes
AWS CloudTrail	AWS::CloudTrail::EventDataStore	No	Yes
AWS CloudTrail	AWS::CloudTrail::Trail	No	Yes
Amazon CloudWatch	AWS::CloudWatch::MetricStream	Yes	No
Amazon CloudWatch Logs	AWS::Logs::LogGroup	Yes	No
Amazon CloudWatch RUM	AWS::RUM::AppMonitor	No	Yes
Amazon CloudWatch Synthetics	AWS::Synthetics::Canary	Yes	No

AWS Service	Resource	Metrics	Logs
AWS CodeBuild	AWS::CodeBuild::Project	Yes	No
Amazon CodeWhisperer	AWS::CodeWhisperer::Customization	No	Yes
Amazon Cognito user pools	AWS::Cognito::UserPool	Yes	Yes
AWS Config	AWS::Config::ConfigRule	No	Yes
Amazon Connect Customer	AWS::Connect::Instance	No	Yes
AWS Database Migration Service	AWS::DMS::ReplicationInstance	Yes	No
AWS Database Migration Service	AWS::DMS::ReplicationTask	Yes	No
AWS DataSync	AWS::DataSync::Agent	Yes	No
AWS DataSync	AWS::DataSync::Task	Yes	Yes
AWS Directory Service	AWS::DirectoryService::MicrosoftAD	No	Yes

AWS Service	Resource	Metrics	Logs
Amazon DynamoDB	AWS::DynamoDB::Table	Yes	No
DynamoDB Accelerator	AWS::DAX::Cluster	Yes	No
Amazon EC2	AWS::EC2::CapacityReservation	Yes	No
Amazon EC2	AWS::EC2::Instance	Yes	No
Amazon EC2	AWS::EC2::FlowLog	Yes	No
Amazon EC2	AWS::EC2::NATGateway	Yes	No
Amazon EC2	AWS::EC2::NetworkInterface	Yes	Yes
Amazon EC2	AWS::EC2::Subnet	Yes	Yes
Amazon EC2	AWS::EC2::TransitGateway	Yes	No
Amazon EC2	AWS::EC2::TransitGatewayAttachment	Yes	Yes

AWS Service	Resource	Metrics	Logs
Amazon EC2	AWS::EC2::VerifiedAccessInstance	No	Yes
Amazon EC2	AWS::EC2::Volume	Yes	No
Amazon EC2	AWS::EC2::VPC	No	Yes
Amazon EC2	AWS::EC2::VPNConnection	Yes	Yes
Amazon EC2 Auto Scaling	AWS::AutoScaling::AutoScalingGroup	Yes	No
AWS Elastic Beanstalk	AWS::ElasticBeanstalk::Environment	Yes	No
Amazon Elastic Container Service	AWS::ECS::Cluster	Yes	Yes
Amazon Elastic Container Service	AWS::ECS::Service	Yes	Yes
Amazon Elastic File System	AWS::EFS::AccessPoint	Yes	No
Amazon Elastic File System	AWS::EFS::FileSystem	Yes	No
Amazon Elastic File System	AWS::EFS::MountTarget	Yes	No

AWS Service	Resource	Metrics	Logs
Amazon Elastic Kubernetes Service	`AWS::EKS::Cluster`	Yes	Yes
Amazon Elastic Kubernetes Service on AWS Fargate		No	Yes
Elastic Load Balancing	`AWS::ElasticLoadBalancing::LoadBalancer`	Yes	No
Elastic Load Balancing V2	`AWS::ElasticLoadBalancingV2::LoadBalancer`	Yes	No
Elastic Load Balancing V2	`AWS::ElasticLoadBalancingV2::TargetGroup`	Yes	No
Amazon ElastiCache	`AWS::ElastiCache::CacheCluster`	Yes	Yes
AWS Elemental MediaConvert	`AWS::MediaConvert::Queue`	Yes	No
AWS Elemental MediaLive		No	Yes
AWS Elemental MediaLive	`AWS::MediaLive::Channel`	Yes	No

AWS Service	Resource	Metrics	Logs
AWS Elemental MediaPackage	AWS::MediaPackage::Channel	Yes	No
AWS Elemental MediaTailor		No	Yes
Amazon EMR	AWS::EMR::Cluster	Yes	Yes
Amazon EventBridge	AWS::Events::Rule	Yes	Yes
Amazon EventBridge Pipes	AWS::Pipes::Pipe	Yes	Yes
AWS Fault Injection Service	AWS::FIS::ExperimentTemplate	No	Yes
Amazon FinSpace	AWS::FinSpace::Environment	No	Yes
Amazon FSx	AWS::FSx::FileSystem	Yes	No
Amazon GameLift Servers	AWS::GameLift::Fleet	Yes	No
AWS Glue	AWS::Glue::Job	No	Yes
AWS Identity Sync	AWS::IdentitySync::Profile	No	Yes

AWS Service	Resource	Metrics	Logs
Amazon Interactive Video Service	`AWS::IVSChat::LoggingConfiguration`	Yes	Yes
AWS IoT	`AWS::IoT::TopicRule`	Yes	Yes
AWS IoT 1-Click	`AWS::IoT1Click::Device`	Yes	No
AWS IoT FleetWise	`AWS::IoTFleetWise::Vehicle`	No	Yes
AWS IoT SiteWise		No	Yes
AWS Key Management Service	`AWS::KMS::Key`	Yes	No
Amazon Managed Service for Apache Flink	`AWS::KinesisAnalytics::Application`	Yes	Yes
Amazon Data Firehose	`AWS::KinesisFirehose::DeliveryStream`	Yes	Yes
Amazon Kinesis Data Streams	`AWS::Kinesis::Stream`	Yes	No
Amazon Kinesis Video Streams	`AWS::KinesisVideo::Stream`	Yes	No

AWS Service	Resource	Metrics	Logs
AWS Lambda	`AWS::Lambda::Function`	Yes	No
Amazon Lex		No	Yes
AWS Mainframe Modernization	`AWS::M2::Application`	No	Yes
Amazon Managed Streaming for Apache Kafka	`AWS::Kafka::Cluster`	Yes	No
Amazon Managed Streaming for Apache Kafka	`AWS::KafkaConnect::Connector`	No	Yes
Amazon Managed Streaming for Apache Kafka	`AWS::MSK::Cluster`	Yes	Yes
Amazon MemoryDB	`AWS::Memorydb::Cluster`	Yes	No
Amazon MQ	`AWS::AmazonMQ::Broker`	Yes	Yes
AWS Network Firewall	`AWS::NetworkFirewall::Firewall`	Yes	Yes
Amazon OpenSearch Service	`AWS::OpenSearchService::Domain`	Yes	No
Amazon OpenSearch Service		No	Yes

AWS Service	Resource	Metrics	Logs
Amazon OpenSearch Service Ingestion	AWS::OSIS::Pipeline	No	Yes
AWS Organizations	AWS::Organizations::Organization	No	Yes
AWS Outposts	AWS::Outposts::Outpost	Yes	No
Amazon Managed Service for Prometheus	AWS::Prometheus::Resource	Yes	No
Amazon Q Business		No	Yes
Amazon QLDB	AWS::QLDB::Ledger	Yes	No
Amazon Quick	AWS::Quicksight::Dashboard	Yes	No
Amazon Quick	AWS::Quicksight::DataSet	Yes	No
Amazon Redshift	AWS::Redshift::Cluster	Yes	Yes
Amazon Redshift Serverless	AWS::RedshiftServerless::Workgroup	Yes	No
Amazon Relational Database Service	AWS::RDS::DBCluster	Yes	Yes

AWS Service	Resource	Metrics	Logs
Amazon Relational Database Service	AWS::RDS::DBInstance	Yes	Yes
Amazon Route 53	AWS::Route53::HealthCheck	Yes	Yes
Amazon Route 53	AWS::Route53::HostedZone	Yes	No
Amazon Route 53 Resolver	AWS::Route53Resolver::ResolverEndpoint	Yes	No
Amazon S3	AWS::S3::Bucket	Yes	No
Amazon SageMaker AI	AWS::SageMaker::Endpoint	Yes	No
Amazon SageMaker AI	AWS::SageMaker::Workteam	No	Yes
AWS Service Catalog	AWS::ServiceCatalog::CloudFormationProduct	Yes	No
Amazon Simple Email Service	AWS::SES::ConfigurationSet	Yes	No

AWS Service	Resource	Metrics	Logs
Amazon Simple Notification Service	AWS::SNS::Topic	Yes	Yes
Amazon Simple Notification Service		No	Yes
Amazon Simple Queue Service	AWS::SQS::Queue	Yes	No
AWS Step Functions	AWS::StepFunctions::Activity	Yes	No
AWS Step Functions	AWS::StepFunctions::StateMachine	Yes	Yes
AWS Storage Gateway	AWS::StorageGateway::Gateway	Yes	No
AWS Storage Gateway	AWS::StorageGateway::Share	No	Yes
AWS Transfer Family	AWS::Transfer::Server	Yes	Yes
Amazon VPC Lattice	AWS::VpcLattice::Service	No	Yes
AWS WAFV2	AWS::WAFv2::WebAcl	No	Yes

AWS Service	Resource	Metrics	Logs
Amazon WorkMail	`AWS::WorkMail::Organization`	Yes	No
Amazon WorkSpaces	`AWS::WorkSpaces::Workspace`	Yes	Yes

How to add related information to custom telemetry sent to CloudWatch

When you publish your own metrics and logs to CloudWatch, the entity information needed for related telemetry is not there by default. When you send metrics to CloudWatch or logs to CloudWatch Logs (with the PutMetricData or PutLogEvents operations), you can add entity information to those logs or metrics. The entity information is associated with the telemetry, and used in the **Explore related** feature to find related telemetry that is associated with the same entity.

The entity sent with telemetry represents a resource or service that the telemetry is associated with. For example, a metric about a service, or that comes from an AWS resource. To identify the entity associated in your code, you create a set of `KeyAttributes` and optional `Attributes` of the entity.

> **Note**
>
> CloudWatch can only find related resources for entities that have had telemetry sent within the previous three hours. If your resource only emits sparse telemetry (less than once every 3 hours), you may want to send additional *heartbeat* telemetry, to keep the entity active within CloudWatch.

For information on how to add entity information, see Entity information in EMF format.

The following sections describe how to create the `KeyAttributes` and `Attributes` so that CloudWatch can identify the resources and services associated with the telemetry.

Identifying the entity with the KeyAttributes object

The `KeyAttributes` property of the Entity objects (in CloudWatch or in CloudWatch logs) uniquely identifies the entity for CloudWatch. It is a list of key-value pairs. Entities with the same `KeyAttributes` are considered to be the same entity. Telemetry associated with the same entity are considered related, and can be easily found in the **Explore related** pane.

> **Note**
>
> In the CloudWatch API, the property is called `KeyAttributes`. In the CloudWatch Logs API, the property is called `keyAttributes`. Here they are treated as the same property.

There are five possible types of object that an `Entity` can represent.

- **AWS::Resource** – The entity represents an AWS resource, such as a DynamoDB table or Amazon EC2 instance.
- **AWS::Service** – The entity represent an AWS service, such as Amazon S3. This might be used, for example, when calling the `ListBuckets` operation, which is not associated with a specific Amazon S3 resource.
- **Service** – The entity represents a workload running in your account. For example, an application or service that you manage.
- **Resource** – The entity represents a resource that is not managed by AWS, for example, operating system resources, such as processes or file volumes.
- **RemoteService** – The entity represents an external service in a remote call. For example, a remote call to a database, external cache, or an external endpoint.

Depending on which of the above types you are trying to represent, you must provide the correct key-value pairs for the `KeyAttributes`. The following describes each type.

AWS::Resource

To specify an AWS Resource, you must include the following three key value pairs:

- `"Type": "AWS::Resource"` – This key-value pair identifies the entity as an AWS resource.
- `"ResourceType": "<resource-type>"` – The string value of the `ResourceType` is the CloudFormation resource type string. For example, `AWS::DynamoDB::Table`.

- `"Identifier": "<resource-id>"` – The primary identifier for the resource. For more information, see primaryIdentifier in the *Extension Development for CloudFormation User Guide*.

AWS::Service

To specify an AWS Service, you must include the following two key value pairs:

- `"Type": "AWS::Service"` – This key-value pair identifies the entity as an AWS service.
- `"Name": "<service-name>"` – The value of the `Name` is the CloudFormation service name string. For example, `AWS::DynamoDB`.

Service

To specify a service that is not operated by AWS, you must include the following three key value pairs:

- `"Type": "Service"` – This key-value pair identifies the entity as a service.
- `"Name": "<service-name>"` – This represents the name of the service that is sending the metrics. For example, `my-service-frontend` or `api.myservice.com`.
- `"Environment": "<environment-name>"` – This attribute specifies where the service is hosted, or the environment to which it belongs. For example `us-west-2`, or `myservice.production`.

Resource

To specify a resource that is not provided by AWS, you must include the following three key value pairs:

- `"Type": "Resource"` – This key-value pair identifies the entity as a resource.
- `"ResourceType": "<resource-type>"` – A string specifying the type of resource. For example, `K8s::Pod` for a Kubernetes Pod.
- `"Identifier": "<resource-id>"` – A string identifier for the resource. Can contain multiple names, separated by pipes. For example, a Kubernetes Pod might be represented by it's cluster name, namespace, and pod name, such as `MyCluster|MyNamespace|MyPod`.

RemoteService

To specify a remote service, you must include the following two key value pairs:

- `"Type": "RemoteService"` – This key-value pair identifies the entity as a remote service.
- `"Name": "<remote-service-name>"` – Specifies how the application refers to the external service in a remote call. For example, `api.test.myservice.com`.

Providing additional details about the entity with the Attributes object

You can provide additional details about the `Entity` that you provide with your telemetry. This can include details about the platform, resource, application, or telemetry provider. The following tables describes the key words that you can use for each of these types of data.

> **Note**
>
> In the CloudWatch API, the property is called `Attributes`. In the CloudWatch Logs API, the property is called `attributes`. Here they are treated as the same property.

Platform details

Keyword	Usage	Domain of values	Examples
`PlatformType`	Defines the hosted-in platform.	AWS::EKS, AWS::ECS, AWS::EC2, AWS::Lambda, K8s, Generic	`AWS::EC2`
`EKS.Cluster`	Name of the Amazon EKS cluster.	Alphanumeric string with basic delimiters.	`FlyingSquad`
`K8s.Cluster`	Name of the self-hosted Kubernetes cluster.	Alphanumeric string with basic delimiters.	`minicube`
`K8s.Namespace`	Name of Kubernete s namespace in	Alphanumeric string with basic delimiters.	`default`, `pet-clinic`

Keyword	Usage	Domain of values	Examples
	Amazon EKS or K8s clusters.		
K8s.Workload	Name of Kubernetes workload in Amazon EKS and K8s clusters.	Alphanumeric string with basic delimiters.	frontend
K8s.Node	Identity of Kubernetes node in Amazon EKS and K8s clusters.	K8s node name (for example, Amazon EC2 instance DNS name).	ip-11-22-33-44.ec2.internal
K8s.Pod	Identity of Kubernetes pod in Amazon EKS and K8s clusters.	K8s pod identifier.	frontend-1234abcd56-ef7890
EC2.AutoScalingGroup	Name of the Amazon EC2 AutoScaling Group.	Alphanumeric string with basic delimiters.	my-asg-name-1
EC2.InstanceId	Identity of the Amazon EC2 instance.	Amazon EC2 instance identifier.	i-1234abcd5678ef90
ECS.Cluster	Identity of the Amazon ECS cluster.	Amazon ECS cluster name.	MyCluster
ECS.Service	Identity of the Amazon ECS service.	Amazon ECS service name.	MyService
ECS.Task	Identity of the Amazon ECS task.	Amazon ECS task ID.	task-123abc
Lambda.Function	Identity of the Lambda function.	Lambda function name.	MyFunction

Keyword	Usage	Domain of values	Examples
`Host`	Name of the host for all platform types.	Sub-domain format.	`ip-111-22-33-44.example.com`

Resource details

Keyword	Usage	Domain of values	Examples
`AWS.Resource.ARN`	ARN for the AWS resource.	Alphanumeric string with basic delimiters.	`arn:aws:dynamodb:us-east-1:123456789012:table/myDynamoDBTable`

Application details

Keyword	Usage	Domain of values	Examples
`AWS.Application`	Name of the application in AppRegistry.	Alphanumeric string with basic delimiters.	`PetClinicApp`
`AWS.Application.ARN`	ARN of the application in AppRegistry.	Alphanumeric string with basic delimiters.	`arn:aws:servicecatalog:us-east-1:1234567890:/applications/...`

Telemetry provider details

Keyword	Usage	Domain of values	Examples
`Telemetry.SDK`	The fingerprint of OTEL SDK version for instrumented services.	Alphanumeric string with basic delimiters.	`opentelemetry,1.32.0-aws-SNAPSHOT,java,Auto`
`Telemetry.Agent`	The fingerprint of the Agent used to collect and send telemetry data.	Alphanumeric string with basic delimiters.	`CWAgent/1.300026.3, ADOTCollector/1.x`
`Telemetry.Source`	Specifies the point of application where the telemetry was collected or what was used for the source of telemetry data.	ServerSpan, ClientSpan, ProducerSpan, ConsumerSpan, LocalRoot Span, JMX, OS.	`ClientSpan, JMX`

Metrics in Amazon CloudWatch

Metrics are data about the performance of your systems. Amazon CloudWatch collects metrics through two paths: AWS vended metrics from services such as Amazon EC2, Amazon EBS, and Amazon RDS, and custom metrics that you publish using the OpenTelemetry Protocol (OTLP) or the CloudWatch API.

Overview

OpenTelemetry: open-source native metrics in CloudWatch

CloudWatch supports OpenTelemetry as the recommended path for metric ingestion and querying. You can use vendor-agnostic, open-source instrumentation to send metrics to CloudWatch – the same OTel SDKs and collectors that work with Prometheus, Grafana, and other backends work with CloudWatch out of the box.

Two metric models

CloudWatch supports two metric models. Both are fully supported – choose based on your needs:

	OpenTelemetry Metrics (Recommended)	CloudWatch Metrics (Classic)
Identity	Metric name, up to 150 labels	Namespace, metric name, up to 30 dimensions
Metric types	Gauge, sum, histogram, exponential histogram	Single values, statistic sets
Ingestion	OTLP endpoint (OTel SDKs, collectors)	PutMetricData API, EMF
Query APIs	PromQL query API	GetMetricData, GetMetricStatistics, ListMetrics, CloudWatch Metrics Insights (SQL)
Alarms	PromQL-based CloudWatch alarms	Standard CloudWatch alarms

	OpenTelemetry Metrics (Recommended)	CloudWatch Metrics (Classic)
Console	CloudWatch Query Studio	CloudWatch Metrics console
Pricing model	Per GB ingested	Per metric per month
Storage	Up to 15 months	Up to 15 months (with automatic rollup)
Metric names	Open-source native	Proprietary (CloudWatch-format)
Best for	New workloads, containers, high-cardinality	Existing integrations, low-cardinality AWS service metrics

GetMetricData, GetMetricStatistics, ListMetrics, and CloudWatch Metrics Insights operate only on CloudWatch Metrics (Classic). Use the PromQL query API for OpenTelemetry Metrics.

Getting started

- **New to CloudWatch metrics?** Start with OpenTelemetry Metrics (Recommended).
- **Already using PutMetricData or EMF?** See CloudWatch Metrics (Classic).
- **Want AWS service metrics in PromQL?** Enable AWS vended metrics in OpenTelemetry format.

OpenTelemetry Metrics (Recommended)

OpenTelemetry is an open-source observability framework that provides vendor-agnostic instrumentation for collecting metrics, logs, and traces from your applications. Amazon CloudWatch supports OpenTelemetry natively, enabling you to use community-standard SDKs and collectors to send metrics directly to CloudWatch without vendor lock-in.

With OpenTelemetry metrics in CloudWatch, you get:

- **No vendor lock-in** – Instrument once with OTel SDKs, send to CloudWatch or any OTel-compatible backend. Switch destinations with a config change, not a code change.

- **Open-source native metric names** – Use community-standard metric names (for example, `container_cpu_usage_seconds_total` from cAdvisor) instead of proprietary names. Existing PromQL dashboards and runbooks work as-is.
- **PromQL querying** – Query with the industry-standard Prometheus Query Language. No proprietary query syntax needed.
- **High cardinality at low cost** – Up to 150 labels per data point with per-GB pricing. No penalty for adding pod names, request IDs, or custom tags.
- **Unified telemetry pipeline** – The same OTel Collector that collects your metrics can also collect logs and traces, simplifying your observability stack.

Topics

- CloudWatch OpenTelemetry Metrics
- Publish custom metrics with OpenTelemetry
- How CloudWatch stores histogram metrics
- Query metrics with PromQL
- Anomaly detection using PromQL
- CloudWatch detailed monitoring for OpenTelemetry metrics
- OTel metrics pricing and storage
- Migrate from Classic to OTel metrics
- AWS vended metrics in OpenTelemetry format

CloudWatch OpenTelemetry Metrics

OpenTelemetry metrics in CloudWatch use the OpenTelemetry Protocol (OTLP) for ingestion and PromQL for querying. This model supports open-source metric names, up to 150 labels per data point, and includes 15 months of storage with no per-metric charges.

Use OpenTelemetry metrics when:

- You are starting new workloads and want the recommended path
- You need high-cardinality labels (pod names, custom tags, OTel semantic conventions)
- You want to query with PromQL (industry-standard, compatible with existing Grafana dashboards)

- You use Container Insights OTel, Application Signals, or AWS vended OTel metrics

In this section

Page	Description
Publish custom metrics with OpenTelemetry	Publish custom metrics through the OTLP endpoint using OTel SDKs or collectors
AWS vended metrics in OpenTelemetry format	Enable AWS services to publish metrics as OTel data points with resource tags. Uses OTel Vended Metric Enrichmen t to convert existing AWS service metrics (Amazon EC2, Amazon RDS, ELB, and so on) into the OTel path with full label enrichment — no agent or code changes required.
Query metrics with PromQL	Query, aggregate, and filter OTel metrics using the Prometheus Query Language
OTel metrics pricing and storage	Per-GB ingestion pricing, included 15-month retention, no per-metric or API charges
Migrate from Classic to OTel metrics	Incremental migration from PutMetricData/EMF to OTLP

Publish custom metrics with OpenTelemetry

You can publish custom metrics to CloudWatch using the OpenTelemetry Protocol (OTLP). You can use OTel SDKs (Java, Python, Go, .NET, Node.js), the OTel Collector, or any OTLP-compatible client.

CloudWatch OTLP endpoint

Send metrics to the CloudWatch OTLP endpoint in your Region:

```
https://monitoring.region.amazonaws.com/v1/metrics
```

Authentication uses standard AWS SigV4 signing. The service name is `monitoring`. For more information about the endpoint, authentication options, and limits, see OTLP Endpoints.

Quick start: publish your first metric

Option 1: OTel Collector (recommended for production)

Configure an OTel Collector with the OTLP HTTP exporter and SigV4 authentication:

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

processors:
  batch:
    send_batch_size: 200
    timeout: 10s

exporters:
  otlphttp:
    tls:
      insecure: false
    metrics_endpoint: "https://monitoring.us-east-1.amazonaws.com/v1/metrics"
    auth:
      authenticator: sigv4auth

extensions:
  sigv4auth:
    service: "monitoring"
    region: "us-east-1"

service:
  extensions: [sigv4auth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]
```

For detailed setup instructions, see Getting started.

Option 2: OTel SDK (Python example)

```
from opentelemetry import metrics
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader
from opentelemetry.exporter.otlp.proto.http.metric_exporter import OTLPMetricExporter

# Point at the CloudWatch OTLP endpoint
exporter = OTLPMetricExporter(
    endpoint="https://monitoring.us-east-1.amazonaws.com:443/v1/metrics"
)
reader = PeriodicExportingMetricReader(exporter, export_interval_millis=60000)
provider = MeterProvider(metric_readers=[reader])
metrics.set_meter_provider(provider)

# Create and record a metric
meter = metrics.get_meter("my-app")
counter = meter.create_counter("http_requests_total", description="Total HTTP
 requests")
counter.add(1, {"method": "GET", "path": "/api/users", "status": "200"})
```

Option 3: OTel SDK (Java example)

```
import io.opentelemetry.api.metrics.Meter;
import io.opentelemetry.api.metrics.LongCounter;

Meter meter = GlobalOpenTelemetry.getMeter("my-app");
LongCounter counter = meter.counterBuilder("http_requests_total")
    .setDescription("Total HTTP requests")
    .build();

counter.add(1, Attributes.of(
    AttributeKey.stringKey("method"), "GET",
    AttributeKey.stringKey("path"), "/api/users",
    AttributeKey.stringKey("status"), "200"
));
```

IAM permissions required

The identity sending metrics needs the `cloudwatch:PutMetricData` permission. For SigV4 authentication, attach the following policy:

```
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:PutMetricData"
  ],
  "Resource": "*"
}
```

For bearer token authentication, see Setting up bearer token authentication for Metrics.

Verify metrics are arriving

Open the CloudWatch console, navigate to **Query Studio**, and run:

```
http_requests_total
```

Metrics typically appear within 1–2 minutes of the first data point being sent.

Supported metric types

The following table describes the OTel metric types that CloudWatch supports and how to query them with PromQL.

OTel metric type	PromQL behavior
Counter	Use `rate()` or `increase()` to query
Gauge	Query directly (current value)
Histogram	Use `histogram_quantile()` for percentiles

Best practices

Follow these recommendations when sending metrics through OTLP:

- **Use meaningful metric names** — Follow OTel naming conventions (for example, `http.server.request.duration` or `http_request_duration_seconds`).
- **Keep label cardinality reasonable** — Avoid request IDs or UUIDs as label values.

- **Set an appropriate export interval** — 60 seconds is standard. Shorter intervals increase cost.
- **Use resource attributes** — Use resource attributes for static metadata (service name, version, environment) rather than per-data-point labels.

How CloudWatch stores histogram metrics

Histogram metrics describe the distribution of many measurements, such as request latency or payload size, instead of a single value. OpenTelemetry defines two histogram types, and Amazon CloudWatch (CloudWatch) stores both as exponential histograms:

- **Explicit-bucket histograms** – also called fixed-boundary histograms. You choose the bucket boundaries in advance, and every measurement is counted into the bucket that contains it. This is the default aggregation for the `Histogram` instrument in most OpenTelemetry SDKs, and it is the format that Prometheus classic histograms use.
- **Exponential histograms** – the bucket boundaries follow an exponential scale rather than a fixed list, so bucket width grows with the magnitude of the value. This gives consistent relative resolution across a wide range of values without you having to choose boundaries.

CloudWatch stores exponential histograms directly. It might reduce the scale of a histogram, which merges adjacent buckets and lowers resolution while preserving the total count and sum. The OpenTelemetry specification allows this, and any consumer of exponential histograms can do it. For more information, see ExponentialHistogram in the OpenTelemetry metrics data model.

When you send an explicit-bucket histogram, CloudWatch converts it to an exponential histogram at ingestion. This is a different operation from a scale reduction: the two formats describe the same distribution with different buckets, so the conversion has to place the counts from each source bucket into the exponential buckets that cover the same range of values.

What the conversion preserves

The following values are carried through the conversion exactly, and queries against them return the same result they would for the source histogram:

- **Count** – the total number of recorded measurements.
- **Sum** – the sum of the recorded values. Because count and sum are both exact, the average is also exact.

- **Minimum and maximum** – when the source histogram carries them. Both fields are optional in OpenTelemetry, and not every SDK and exporter records them, so they aren't always available. When they are missing, CloudWatch doesn't estimate them.

Percentiles are the exception. A percentile read from a histogram of any kind is an estimate, because the histogram records how many measurements fell in each bucket but not the individual values. That is true of the source explicit-bucket histogram before any conversion takes place.

How counts are redistributed

To place a source bucket's count into exponential buckets, CloudWatch subdivides the range that the source bucket covers. It then distributes the count across those subdivisions, weighting them by how the density of measurements compares with neighboring buckets instead of spreading them evenly.

> **Note**
>
> Estimating a percentile directly from an explicit-bucket histogram – for example, with `histogram_quantile()` in Prometheus – interpolates linearly within the bucket that contains the percentile. This approach assumes that measurements are spread evenly across that bucket. Long-tailed distributions such as latency usually violate that assumption, so this is a known source of error in percentile estimates from explicit-bucket histograms. For more information, see `histogram_quantile()` in the Prometheus documentation.

What limits percentile accuracy

How closely an estimated percentile matches the true value depends mostly on the source histogram, not on the conversion. Once measurements are counted into a bucket, their individual values are gone, and no conversion can recover them. Accuracy is therefore best when your bucket boundaries are narrow across the range where your measurements actually fall.

Estimates are least accurate in these cases:

- **Wide buckets** – a percentile that falls inside a wide bucket can only be located somewhere within that bucket's range.

- **The unbounded top bucket** – the last bucket of an explicit-bucket histogram has no upper boundary. High percentiles that fall into this bucket are therefore the least constrained. This matters most for tail latency, where the values you care about are the ones above your highest boundary. If your SDK can record a minimum and maximum, enable it: the bucket counts alone don't capture the true extremes, and CloudWatch preserves those two values when they're present.
- **Boundaries that don't match your data** – if most measurements land in one or two buckets, the histogram carries little information about the distribution regardless of how many buckets you defined.

If you rely on percentiles, configure your OpenTelemetry SDK to use exponential bucket aggregation for your `Histogram` instruments instead of the default explicit buckets. This avoids the redistribution described earlier, and you don't have to predict the range of your measurements in advance. For more information, see Base2 Exponential Bucket Histogram Aggregation in the OpenTelemetry documentation.

Querying histogram metrics

You query histograms in CloudWatch with the same PromQL histogram functions regardless of which format you sent. To estimate a percentile, use `histogram_quantile()`; to get the exact average, use `histogram_avg()`. For example, the following query returns the estimated 95th percentile of request duration:

```
histogram_quantile(0.95, http_request_duration_seconds)
```

For more information about querying metrics with PromQL, see PromQL querying.

Query metrics with PromQL

Topics

- What is Prometheus Query Language (PromQL)?
- PromQL limits and restrictions
- Supported AWS Regions
- IAM permissions for PromQL
- PromQL querying
- Grafana integration

- Running PromQL queries in Query Studio
- Using PromQL in alarms
- Prometheus-compatible APIs

What is Prometheus Query Language (PromQL)?

Prometheus Query Language (PromQL) is a functional query language that lets you select, aggregate, and transform time series data in real time. PromQL was originally designed for Prometheus and has become a popular query language for metrics.

Amazon CloudWatch supports PromQL for querying metrics including metrics ingested through OpenTelemetry Protocol (OTLP) and AWS enriched vended metrics. When you ingest OTLP metrics, CloudWatch preserves the full semantic structure of your telemetry, including resource attributes, instrumentation scope, datapoint attributes, and AWS-specific metadata, and exposes them as queryable PromQL labels.

With PromQL you can do the following:

- Select time series by metric name and label matchers.
- Apply mathematical functions and operators across time series.
- Aggregate metrics across dimensions such as service, region, or account.
- Compute rates, histograms, quantiles, and moving averages.

You can use PromQL queries interactively in Running PromQL queries in Query Studio and also to create CloudWatch Alarms. For more information, see PromQL querying and Using PromQL in alarms.

> **Note**
>
> CloudWatch uses PromQL based on the Prometheus 3.0 specification. This includes support for UTF-8 metric names and label names.
> Some query patterns from Prometheus documentation assume cumulative histograms and don't apply to OpenTelemetry delta histograms. For more information, see Querying histogram metrics.

The following concepts are fundamental to working with PromQL in CloudWatch.

Concept	Description
Time series	A stream of timestamped values identified by a metric name and a set of key-value pairs called *labels*. Each unique combination of metric name and labels forms a distinct time series.
Instant vector	A set of time series containing a single sample for each series, all sharing the same timestamp. Returned by queries like `{"http.server.active_requests", "@resource.service.name"="myservice"}` .
Range vector	A set of time series containing a range of data points over time for each series. Created by appending a time duration selector in brackets, for example, `avg_over_time({"http.server.active_requests", "@resource.service.name"="myservice"}[5m])`.
Label	A key-value pair attached to a time series. In OTLP-ingested metrics, labels are derived from resource attributes, instrumentation scope, datapoint attributes, and AWS-specific metadata.
Label matcher	An expression in curly braces that filters time series by label value. Supports exact match (=), not equal (!=), regex match (=~), and negative regex match (!~).
Aggregation operator	A function that combines multiple time series into fewer series. Common operators include `sum`, `avg`, `min`, `max`, `count`, and `topk`.

PromQL limits and restrictions

The following table lists the limits and restrictions for PromQL:

Limit	Value	Additional information	Error code
Max TPS for query requests per account	300	Maximum number of query requests (/query, /query_ra	422

Limit	Value	Additional information	Error code
		nge) per second allowed per account.	
Max TPS for discovery requests per account	10	Maximum number of discovery requests (/series, /label, /label_values) per second allowed per account.	422
Max concurrent query requests per account	30	Maximum number of queries (/query, /query_range) an account can have actively executing at the same time.	429
Max concurrent discovery requests per account	30	Maximum number of discovery requests (/series, /labels, /label_values) an account can have actively executing at the same time.	429
Max series returned per query request	500	Maximum number of unique time series a query request (/query, /query_range) can return.	200 - truncated response
Max labels returned per discovery request	10,000	Maximum number of unique labels a discovery request (/series, /labels, /label_values) can return.	200 - truncated response
Max range per request	7 days	Maximum time range a query can span, including range parameters and lookback periods.	422

Limit	Value	Additional information	Error code
Max series scanned per 24h window	100,000	Maximum number of unique time series that can be scanned per 24-hour window of query execution.	422
Max samples scanned per 24h window	300,000,000	Maximum number of samples that can be scanned per 24-hour window of query execution.	422
Max samples processed per 24h window	3,000,000,000	Maximum number of samples that can be processed per 24-hour window of query execution.	422
Execution timeout	20 seconds	Maximum time the engine can spend evaluating a query, excluding time spent in queue and fetching data from storage.	422

Supported AWS Regions

The following table lists the AWS Regions where OTLP metrics ingestion, PromQL querying, and Query Studio are available.

Region name	Region code	OTLP metrics ingest	PromQL query	Query Studio
US East (N. Virginia)	us-east-1	✓	✓	✓
US West (Oregon)	us-west-2	✓	✓	✓
Europe (Ireland)	eu-west-1	✓	✓	✓

Region name	Region code	OTLP metrics ingest	PromQL query	Query Studio
Asia Pacific (Singapore)	ap-southeast-1	✓	✓	✓
Asia Pacific (Sydney)	ap-southeast-2	✓	✓	✓
US West (N. California)	us-west-1	✓	✓	✓
US East (Ohio)	us-east-2	✓	✓	✓
Europe (London)	eu-west-2	✓	✓	✓
Europe (Paris)	eu-west-3	✓	✓	✓
Europe (Frankfurt)	eu-central-1	✓	✓	✓
Europe (Stockholm)	eu-north-1	✓	✓	✓
Europe (Milan)	eu-south-1	✓	✓	✓
Europe (Spain)	eu-south-2	✓	✓	✓
Europe (Zurich)	eu-central-2	✓	✓	✓
Asia Pacific (Tokyo)	ap-northeast-1	✓	✓	✓
Asia Pacific (Seoul)	ap-northeast-2	✓	✓	✓
Asia Pacific (Osaka)	ap-northeast-3	✓	✓	✓
Asia Pacific (Mumbai)	ap-south-1	✓	✓	✓
Asia Pacific (Hyderabad)	ap-south-2	✓	✓	✓

Region name	Region code	OTLP metrics ingest	PromQL query	Query Studio
Asia Pacific (Hong Kong)	ap-east-1	✓	✓	✓
Asia Pacific (Taipei)	ap-east-2	✓	✓	✓
Asia Pacific (Jakarta)	ap-southeast-3	✓	✓	✓
Asia Pacific (Melbourne)	ap-southeast-4	✓	✓	✓
Asia Pacific (Malaysia)	ap-southeast-5	✓	✓	✓
Asia Pacific (Auckland)	ap-southeast-6	✓	✓	✓
Asia Pacific (Thailand)	ap-southeast-7	✓	✓	✓
South America (São Paulo)	sa-east-1	✓	✓	✓
Canada (Central)	ca-central-1	✓	✓	✓
Canada West (Calgary)	ca-west-1	✓	✓	✓
Africa (Cape Town)	af-south-1	✓	✓	✓
Mexico (Central)	mx-central-1	✓	✓	✓

IAM permissions for PromQL

To execute PromQL queries, you need both `cloudwatch:GetMetricData` and `cloudwatch:ListMetrics` permissions. The following table lists the Prometheus-compatible API endpoint paths and their required IAM actions:

API endpoint path	HTTP methods	Required actions
`/api/v1/query`	GET, POST	`cloudwatch:GetMetricData` , `cloudwatch:ListMetrics`
`/api/v1/query_range`	GET, POST	`cloudwatch:GetMetricData` , `cloudwatch:ListMetrics`
`/api/v1/series`	GET, POST	`cloudwatch:ListMetrics`
`/api/v1/labels`	GET, POST	`cloudwatch:ListMetrics`
`/api/v1/label/` *`label_name`* `/values`	GET	`cloudwatch:ListMetrics`

PromQL querying

When you ingest OpenTelemetry metrics into CloudWatch through the Metrics endpoint, the hierarchical OTLP data model is flattened into PromQL-compatible labels. This section describes the label structure, the PromQL syntax for querying these labels, and the UTF-8 support in PromQL.

> **Note**
>
> PromQL in Prometheus 3 supports full UTF-8 characters in metric names and label names. This is particularly important for OTLP metrics, because OpenTelemetry semantic conventions use dots in attribute names such as `service.name`. Previously, these dots were replaced with underscores during translation, causing discrepancies between what was defined in OTel conventions and what was queryable in Prometheus.
> A metric name that contains dots must be quoted, either inside the curly braces, as in `{"http.server.active_requests"}`, or as the value of the `__name__` label, as in `{__name__="http.server.active_requests"}`. A bare, unquoted name such as `http.server.active_requests` isn't valid PromQL.

When using PromQL in CloudWatch, the @ prefix convention distinguishes OTLP-scoped labels from standard Prometheus labels. Fields within each scope use a double-@ prefix (for example, `@resource.@schema_url`), while attributes use a single-@ scope prefix, for example, `@resource.service.name`. Datapoint attributes also support bare (un-prefixed) access for backward compatibility with standard PromQL queries, for example, `{"http.server.active_requests"}` and `{"@datapoint.@name"="http.server.active_requests"}` are equivalent.

A PromQL expression is enclosed in curly braces, specifying the metric name and an optional set of label matchers. The following example selects all time series for the `http.server.active_requests` metric:

```
{"http.server.active_requests"}
```

The following example selects all time series for the metric `http.server.active_requests` where the OpenTelemetry resource attribute `service.name` equals `myservice`:

```
{"http.server.active_requests", "@resource.service.name"="myservice"}
```

You can combine multiple label matchers in a single query. The following example selects all time series for the `http.server.active_requests` metric where the OpenTelemetry resource attribute `service.name` equals `myservice` across all US regions:

```
{"http.server.active_requests",
 "@resource.service.name"="myservice",
 "@aws.region"=~"us-.*"}
```

The following example shows a range query. It calculates the average value of all datapoints within a specified time range for each time series:

```
avg_over_time(
  {"http.server.active_requests",
   "@resource.service.name"="myservice"}[5m]
)
```

The following table summarizes the prefix conventions for each OTLP scope:

OTLP scope	Fields prefix	Attributes prefix	Example
Resource	`@resource.@`	`@resource.`	`@resource.service.name="myservice"`
Instrumentation Scope	`@instrumentation.@`	`@instrumentation.`	`@instrumentation.@name="otel-go/metrics"`
Datapoint	`@datapoint.@`	`@datapoint.` or bare	`cpu="cpu0"` or `@datapoint.cpu="cpu0"`
AWS-reserved	N/A	`@aws.`	`@aws.account_id="123456789"`

Querying vended AWS metrics with PromQL

To be able to query vended AWS metrics in PromQL, you first need to enable OTel enrichment of vended metrics. See: AWS vended metrics in OpenTelemetry format.

After you enable OTel enrichment, vended AWS metrics become queryable through PromQL with additional labels. The metric name is the same as the original CloudWatch metric name, and the original CloudWatch dimensions are available as datapoint attributes. The following labels are available (the example below is for an EC2 instance):

PromQL Label	Description	Example
`InstanceId`	Original CloudWatch dimension, as a datapoint attribute	`i-0123456789abcdef0`
`"@resource.cloud.resource_id"`	Full ARN of the resource	`arn:aws:ec2:us-east-1:123456789012:instance/i-0123456789abcdef0`
`"@resource.cloud.provider"`	Cloud provider	`aws`

PromQL Label	Description	Example
"@resource.cloud.region"	AWS Region where this metric originated	us-east-1
"@resource.cloud.account.id"	AWS account ID where this metric originated	123456789012
"@instrumentation.@name"	Instrumentation scope name identifying the source service	cloudwatch.aws/ec2
"@instrumentation.cloudwatch.source"	Source service identifier	aws.ec2
"@instrumentation.cloudwatch.solution"	Enrichment solution identifier	CloudWatchOTelEnrichment
"@aws.tag.Environment"	AWS resource tag	production
"@aws.account"	AWS account where this metric was ingested (system label)	123456789012
"@aws.region"	AWS Region where this metric was ingested (system label)	us-east-1

The following example selects `Invocations` for a specific Lambda function:

```
{Invocations, FunctionName="my-api-handler"}
```

The following example selects Lambda `Errors` for all functions tagged with a specific team:

```
{Errors, "@instrumentation.@name"="cloudwatch.aws/lambda", "@aws.tag.Team"="backend"}
```

The following example computes the total Lambda `Invocations` grouped by team:

```
sum by ("@aws.tag.Team")(
    {Invocations, "@instrumentation.@name"="cloudwatch.aws/lambda"}
)
```

The following example selects all time series for the EC2 `CPUUtilization` metric. The usage of `"@instrumentation.@name"="cloudwatch.aws/ec2"` is to exclusively match CPUUtilization from EC2 and not from other AWS services such as Amazon Relational Database Service:

```
histogram_avg({CPUUtilization, "@instrumentation.@name"="cloudwatch.aws/ec2"})
```

Querying histogram metrics

OpenTelemetry histogram metrics, such as request durations and latencies, are stored as histogram samples rather than float samples. Some AWS vended metrics are also stored as histograms. That's why the previous example uses `histogram_avg`. Histogram samples behave differently from float samples in PromQL, so a query pattern that works for one type can return an empty or incorrect result for the other.

To check whether a metric is a histogram, select its raw time series and inspect the `__type__` label on the results:

```
{"http.server.request.duration"}
```

Delta and cumulative histograms

OpenTelemetry histograms use one of two aggregation temporalities:

Cumulative

Each datapoint is a running total since the start of the series.

Delta

Each datapoint is a complete total for its own export period.

The temporality determines which query pattern returns a correct result. CloudWatch reports a histogram's temporality in the `__temporality__` label, whose value is either `cumulative` or `delta`. To check the temporality of a metric, select its raw time series and inspect the `__temporality__` label on the results:

```
{"http.server.request.duration"}
```

The `rate`, `increase`, `irate`, and `resets` functions assume cumulative data. They measure growth between datapoints. CloudWatch applies this assumption to every histogram, including delta histograms. On a delta histogram, these functions return an incorrect result without a warning. For example, the common Prometheus pattern `histogram_count(rate(`*`histogram`*`[5m]))` returns the correct observation rate for a cumulative histogram. It doesn't return a correct rate for a delta histogram.

For a cumulative histogram, use `rate`. The following example returns the observation rate, in observations per second:

```
sum(histogram_count(rate({"http.server.request.duration"}[5m])))
```

For a delta histogram, add up the per-period totals with `sum_over_time` and divide by the length of the range. This example also returns the observation rate:

```
sum(histogram_count(sum_over_time({"http.server.request.duration"}[5m]))) / 5m
```

You can adapt the delta pattern as follows:

- To return the total number of observations in the range instead of a rate, remove `/ 5m`.
- To return the mean observed value in the range, use `histogram_avg` instead of `histogram_count`.

> **Note**
>
> Don't use a subquery such as `rate(histogram_count(`*`histogram`*`)[5m:1m])` to calculate a rate from a delta histogram. It returns the correct result only when the export interval equals the subquery step. If the export interval changes, it returns an incorrect result without a warning.

Getting statistics from a histogram

The `min` and `max` aggregation operators operate only on float samples. They skip histogram samples, so applying them to a histogram metric returns an empty result rather than an error. The response includes an informational annotation that contains the text `ignored histogram`.

To read a statistic from a histogram metric, use a histogram function:

`histogram_quantile(`*`q`*`, `*`histogram`*`)`

Returns the value at the *q* quantile — the value below which a fraction *q* of observations fall. *q* is a value between 0 and 1. For example, use `0.99` for p99, not 99.

`histogram_avg(`*`histogram`*`)`

Returns the mean of the observed values.

`histogram_count(`*`histogram`*`)` and `histogram_sum(`*`histogram`*`)`

Return the number of observations and the sum of the observed values.

PromQL doesn't provide an exact maximum for a histogram. The closest equivalent is a high quantile such as `histogram_quantile(1, `*`histogram`*`)`. Because a quantile is estimated from bucket boundaries, the result can be slightly higher than the largest observed value.

For a delta histogram, apply the function directly to the metric. The following example returns the p99 duration for each export period:

```
histogram_quantile(0.99, {"http.server.request.duration"})
```

The preceding form returns the p99 for a single export period. To calculate the p99 over a longer window, first combine the per-period histograms with `sum_over_time`. You don't divide by the range, because a quantile is unaffected by scaling the bucket counts. The following example returns the p99 duration over the last five minutes:

```
histogram_quantile(0.99, sum_over_time({"http.server.request.duration"}[5m]))
```

For a cumulative histogram, applying the function directly returns a statistic over the whole life of the series. To get a statistic for a recent window, apply the function to `rate`. The following example returns the p99 duration over the last five minutes:

```
histogram_quantile(0.99, rate({"http.server.request.duration"}[5m]))
```

Keeping labels in the result

An aggregation operator such as `avg` or `sum` combines all matching time series into one. It removes every label that you don't group by. To keep one result per label value, add a `by` clause. The following example returns the observation rate of a delta histogram for each service:

```
sum by ("@resource.service.name")(
    histogram_count(sum_over_time({"http.server.request.duration"}[5m]))
) / 5m
```

Histogram functions such as `histogram_quantile` and `histogram_avg` are functions rather than aggregations, so they keep all labels and return one result for each time series.

Querying with MCP tools

The CloudWatch MCP Server provides Model Context Protocol (MCP) tools that let AI assistants and development tools query CloudWatch PromQL data on your behalf. The MCP tools handle authentication and request formatting automatically, so you can focus on writing PromQL queries rather than managing HTTP requests and SigV4 signing.

The following PromQL tools are available in the CloudWatch MCP Server:

Tool	Description
`execute_promql_query`	Runs an instant PromQL query, returning metric values at a single point in time.
`execute_promql_range_query`	Runs a PromQL range query over a time window, returning time series data for trend analysis and graphing.
`get_promql_label_values`	Retrieves values for a specific PromQL label, such as `__name__` for metric names or `@resource.service.name` for services.
`get_promql_series`	Finds time series matching PromQL label selectors and returns the full label set of each matching series.

Tool	Description
`get_promql_labels`	Lists all available PromQL label names to help discover the label structure of your metrics.

For full details on parameters, configuration, and setup instructions, see Tools for CloudWatch PromQL in the CloudWatch MCP Server documentation.

Querying with the HTTP API

You can also query CloudWatch PromQL data programmatically by calling the Prometheus-compatible HTTP endpoints directly. Requests must be signed with AWS Signature Version 4 using `monitoring` as the service name.

The PromQL endpoint follows the pattern `https://monitoring.`*`AWS Region`*`.amazonaws.com/api/v1/`*`operation`*. For example, for the US East (N. Virginia) (us-east-1) Region, the endpoint for an instant query is `https://monitoring.us-east-1.amazonaws.com/api/v1/query`.

For the full API reference, including supported operations, request parameters, and response formats, see Prometheus-compatible APIs. For the list of AWS Regions where PromQL querying is available, see Supported AWS Regions. For the IAM actions required for each operation, see IAM permissions for PromQL.

Grafana integration

After you ingest OpenTelemetry metrics into CloudWatch and make them queryable with PromQL, you can visualize and explore that data in Grafana and Amazon Managed Grafana. We recommend that you use the **PromQL** query type in the **Amazon CloudWatch** data source plugin. It uses the same authentication and Region settings as your other CloudWatch queries, so you don't need to configure a separate data source. For more information, see Querying with the CloudWatch data source.

Alternatively, you can connect to the CloudWatch monitoring endpoint through the **Amazon Managed Service for Prometheus** data source plugin, which signs requests with Signature Version 4 (SigV4). Use this option if your Grafana version or Amazon Managed Grafana workspace doesn't support the PromQL query type in the CloudWatch data source. For more information, see Querying from Grafana with the Amazon Managed Service for Prometheus data source and

Querying from Amazon Managed Grafana with the Amazon Managed Service for Prometheus data source.

Querying with the CloudWatch data source

The **Amazon CloudWatch** data source plugin includes a **PromQL** query type, alongside the **Metric Search** and **Metric Insights** query types. The plugin sends SigV4-signed requests to the CloudWatch PromQL endpoint for the selected Region. You don't need to change the data source configuration or set a signing service.

The PromQL query type is available in the following environments:

- **Amazon Managed Grafana** — workspaces that use Grafana version 13 or later. For more information, see Grafana version 13 in the *Amazon Managed Grafana User Guide*.
- **Grafana** — version 12.10.0 or later of the CloudWatch data source plugin, which includes both the Builder and Code editing modes. The plugin requires Grafana `>=11.6.11 <12 || >=12.0.10 <12.1 || >=12.1.7 <12.2 || >=12.2.5`. To update the plugin, use the Grafana plugins catalog.

IAM prerequisites — the IAM principal that the CloudWatch data source uses, such as the Amazon Managed Grafana workspace IAM role, must have both `cloudwatch:GetMetricData` (required for instant and range queries) and `cloudwatch:ListMetrics` (required for metric, label key, and label value autocomplete). For details, see IAM permissions for PromQL.

> **Note**
>
> If the IAM principal has only these permissions, choosing **Save & test** on the CloudWatch data source reports an error for the CloudWatch Logs check because it can't call `logs:DescribeLogGroups`. The metrics check still succeeds, and PromQL queries aren't affected. To clear the error, also grant the permissions that the CloudWatch data source needs for logs queries.

To query CloudWatch metrics with PromQL, complete the following steps.

1. Add an **Amazon CloudWatch** data source, or use an existing one. In Amazon Managed Grafana, you can use the AWS data source configuration option to add the data source and manage its credentials. For more information, see Connect to an Amazon CloudWatch data source in the *Amazon Managed Grafana User Guide*.

2. In a dashboard panel or in **Explore**, select the CloudWatch data source.
3. In the query editor, choose **PromQL** as the query type.
4. For **Region**, keep the data source default or choose the AWS Region that contains your OpenTelemetry metrics.
5. Build the query in one of the following editing modes. Use the **Builder/Code** toggle to switch between them.
   - **Builder** — select a metric, add label filters, and optionally add operations such as aggregations and functions.
   - **Code** — enter a PromQL expression directly. The editor suggests metric names, label keys, and label values. To build a selector visually, choose **Metrics browser**.
6. (Optional) Under the query options, set **Legend**, **Min step**, **Format** (`Time series` or `Table`), and **Type** (`Range` or `Instant`).

Querying from Grafana with the Amazon Managed Service for Prometheus data source

You can also query CloudWatch PromQL data from Grafana by adding the **Amazon Managed Service for Prometheus** data source plugin and pointing it at the CloudWatch monitoring endpoint. SigV4 signing is built in to the plugin and is always enabled, so there is no toggle to turn on. The plugin is published at grafana.com/grafana/plugins/grafana-amazonprometheus-datasource/; install it from the Grafana plugins catalog before adding the data source. AMP plugin v3.0.0 requires Grafana `>=11.6.11 <12 || >=12.0.10 <12.1 || >=12.1.7 <12.2 || >=12.2.5`.

IAM prerequisites — the IAM principal whose credentials Grafana uses must have both `cloudwatch:GetMetricData` (required for instant and range queries) and `cloudwatch:ListMetrics` (required for series and label discovery). For details, see IAM permissions for PromQL.

To configure Grafana, complete the following steps.

1. Install the **Amazon Managed Service for Prometheus** data source plugin from the Grafana plugins catalog.
2. In Grafana, go to **Connections**, **Data sources**, choose **Add data source**, and select **Amazon Managed Service for Prometheus**.
3. Set the data source **URL** to `https://monitoring.`*`AWS Region`*`.amazonaws.com`.

4. Set the **Region** to your AWS Region. Choose an **Authentication provider** appropriate for your environment (default credential chain, access keys, or workspace IAM role).
5. Set the **Service** field to `monitoring`. The plugin defaults this field to `aps` for Amazon Managed Service for Prometheus, but the CloudWatch PromQL endpoint requires `monitoring`.
6. Choose **Save & test**.

> ⚠ **Service must be set to monitoring**
>
> If you leave **Service** at the default value `aps`, every query fails with HTTP 403. The response includes the message `Credential should be scoped to correct service: 'monitoring'`. When you provision the data source as YAML or Terraform, set the equivalent key `sigv4Service` to `monitoring`.

Querying from Amazon Managed Grafana with the Amazon Managed Service for Prometheus data source

If your Amazon Managed Grafana workspace uses Grafana version 12, you can query CloudWatch PromQL data by adding an **Amazon Managed Service for Prometheus** data source that points at the CloudWatch monitoring endpoint. For workspaces that use Grafana version 13 or later, we recommend the CloudWatch data source instead. For more information, see Querying with the CloudWatch data source.

This data source plugin signs requests with SigV4 using the workspace IAM role automatically; SigV4 is always enabled, with no toggle to configure. The plugin is available in Amazon Managed Grafana version 12 and later. For more information, see Connect to an Amazon Managed Service for Prometheus data source in the *Amazon Managed Grafana User Guide*.

IAM prerequisites — the Amazon Managed Grafana workspace IAM role must have both `cloudwatch:GetMetricData` (required for instant and range queries) and `cloudwatch:ListMetrics` (required for series and label discovery). For details, see IAM permissions for PromQL.

To configure the data source, complete the following steps.

1. In your Amazon Managed Grafana workspace, add an **Amazon Managed Service for Prometheus** data source.
2. Set the data source **URL** to `https://monitoring.`*`AWS Region`*`.amazonaws.com`.

3. Set the **Region** to your AWS Region. Amazon Managed Grafana injects credentials from the workspace IAM role automatically; you do not need to configure static keys.
4. Set the **Service** field to `monitoring`. The plugin defaults this field to `aps`, but the CloudWatch PromQL endpoint requires `monitoring`.
5. Choose **Save & test**.

Running PromQL queries in Query Studio

Query Studio is an interactive query environment in the CloudWatch console where you can write, run, and visualize PromQL queries against your CloudWatch metrics. You can use Query Studio to explore metrics ingested through OTLP and AWS vended metrics, create visualizations, set up alarms, and add widgets to your CloudWatch dashboards.

You can run PromQL queries programmatically using the CloudWatch API, or interactively in Query Studio.

Running a PromQL query in Query Studio

To run a PromQL query in Query Studio

1. Open the CloudWatch console.
2. In the navigation pane, choose **Query Studio**.
3. In the query editor tool, select **PromQL** from the drop-down menu.
4. To build the query in **Builder** mode, do the following:

    a. Select a metric name.

    b. (Optional) Select one or more labels and values to filter the time series that the query returns.

    c. (Optional) To aggregate the selected time series, choose the **More options** icon (three dots), and then choose **Add aggregation function**. Then choose a function such as `sum`, `avg`, `min`, or `max`.

> **Note**
>
> An aggregation function applies to all of the time series that match your metric and label selection, and combines them into a single time series. Labels that you don't

group by are removed from the result. To return one result per label value, such as one per host, group by that label.
The `min` and `max` functions operate only on float samples. They skip histogram samples, which are common for durations and latencies, so the query returns an empty result. To get a statistic from a histogram metric, use a histogram function such as `histogram_quantile` or `histogram_avg`. For more information, see Querying histogram metrics.

5. Or enter your PromQL query through the **Editor** mode, for example `{"http.server.active_requests"}`.
6. (Optional) Adjust the time range using the time interval selector at the top of the page.
7. Choose **Run** to execute the query and view the results.

Query Studio will display the results as a time series graph. You can switch between graph and table views to analyze your data.

Creating alarms from Query Studio

After running a PromQL query that returns a single time series, you can create a CloudWatch Alarm directly from Query Studio. Choose **Create alarm** from the actions menu to configure alarm thresholds, evaluation periods, and notification actions based on your query results. For more information, see Using PromQL in alarms.

Adding visualizations to dashboards

You can add any Query Studio visualization to a CloudWatch dashboard. After running a query and viewing the results, choose **Add to dashboard** to save the visualization as a dashboard widget. The widget continues to run the PromQL query at the dashboard's refresh interval, keeping your dashboard up to date.

Using PromQL in alarms

You can create CloudWatch alarms that use PromQL queries to monitor your metrics. PromQL alarms evaluate a PromQL expression and trigger alarm state changes based on the query results.

For information about alarm concepts, see Concepts.

For information about alarm data queries, see Alarm data queries.

For information about alarm actions, see Alarm actions.

For information about alarm limits, see Limits.

Creating a PromQL alarm

You can create a PromQL alarm from the CloudWatch console, the AWS CLI, or the CloudWatch API.

To create a PromQL alarm from the console

1. Open the CloudWatch console.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**, then choose the **PromQL** tab.
5. Enter your PromQL query. The query must return a single time series for the alarm to evaluate.
6. Configure the alarm conditions, including the threshold, evaluation period, and datapoints to alarm.
7. Configure the alarm actions, such as Amazon SNS notifications.
8. Enter a name and description for the alarm, then choose **Create alarm**.

You can also create a PromQL alarm directly from Running PromQL queries in Query Studio after running a query that returns a single time series.

Creating a CloudWatch alarm using PromQL for anomaly detection

You can create a PromQL alarm that triggers when a metric breaches an expected range defined by statistical bounds. The alarm query combines upper and lower bounds into a single expression that returns only the anomalous data points. Any time series returned by the query is considered breaching.

The following example expression detects when an ad request metric exceeds 3 standard deviations from the median over a 60-minute window:

```
1 * {"app.ads.ad_requests"} > quantile_over_time(0.5, {"app.ads.ad_requests"}[60m]
 offset 1m)
    + 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m)
or
1 * {"app.ads.ad_requests"} < clamp_min(
    quantile_over_time(0.5, {"app.ads.ad_requests"}[60m] offset 1m)
    - 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m),
```

```
0)
```

This expression works across multiple label values, so the alarm can track anomalies across your entire fleet. Each breaching time series is tracked as a separate contributor. For more information about how PromQL alarms evaluate contributors, see PromQL alarms.

You can adjust the multiplier and time window to match your metric's behavior. A higher multiplier produces wider bounds with fewer false positives. A longer time window smooths out short-term spikes. The `clamp_min` function prevents the lower bound from going negative for metrics that can't have negative values.

For more information about building anomaly detection bands with PromQL, see Anomaly detection using PromQL.

Prometheus-compatible APIs

Topics

- QueryMetrics
- GetSeries
- GetLabels

Amazon CloudWatch supports the following Prometheus-compatible APIs for querying PromQL data. For information about endpoint URLs, signing requests with Signature Version 4, and required IAM permissions, see PromQL querying.

For the limits and restrictions that apply to all PromQL API requests, including per-account TPS, concurrency, and 24-hour scan windows, see PromQL limits and restrictions.

QueryMetrics

The `QueryMetrics` operation evaluates a PromQL instant query at a single point in time, or evaluates a query over a range of time.

Valid HTTP verbs

`GET, POST`

Valid URIs

`/api/v1/query` — evaluates an instant query at a single point in time.

`/api/v1/query_range` — evaluates a query over a range of time.

The full request URL combines the CloudWatch monitoring host for the AWS Region with the operation path, for example `https://monitoring.AWS Region.amazonaws.com/api/v1/query`. For information about endpoints, signing, and required IAM permissions, see PromQL querying.

URL query parameters

The following parameters are passed in the URL query string for GET requests, or as form-encoded body fields for POST requests.

Parameter	Applies to	Description
`query`	Both	Required. The PromQL expression string to evaluate. The expression must select a metric by name — for example, `{"http.server.active_requests"}` or `{__name__="http.server.active_requests"}` . Selecting series only by labels, without a metric name, is not supported.
`time`	`/api/v1/query`	Optional. Evaluation timestamp as an RFC 3339 timestamp or a Unix timestamp. Defaults to the current server time.
`start`	`/api/v1/query_range`	Required. Start of the time range, as an RFC 3339 timestamp or a Unix timestamp.
`end`	`/api/v1/query_range`	Required. End of the time range, as an RFC 3339 timestamp or a Unix timestamp.
`step`	`/api/v1/query_range`	Required. Query resolution step width, in `duration` format or as a floating-point number of seconds.
`limit`	Both	Optional. Maximum number of unique time series to return, from 1 to 500. If you do not specify `limit`, CloudWatch returns up to the maximum (500). See Result limit.

Duration

A `duration` in a Prometheus-compatible API is a number, followed immediately by one of the following units:

- `ms` milliseconds
- `s` seconds
- `m` minutes
- `h` hours
- `d` days, assuming a day always has 24h
- `w` weeks, assuming a week always has 7d
- `y` years, assuming a year always has 365d

Required IAM permissions

To call `QueryMetrics`, the calling identity must have both of the following IAM actions:

- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`

For the complete IAM action mapping for all PromQL operations, see IAM permissions for PromQL.

Result limit

A single `/api/v1/query` or `/api/v1/query_range` response can return up to **500 unique time series**. To request fewer results, pass the `limit` parameter with a value from `1` to `500`. If you do not specify `limit`, CloudWatch returns up to the maximum.

When the matching series exceed the cap, the response is truncated and a message is included in the standard Prometheus `warnings` field. The HTTP status code remains `200`.

For the full list of PromQL limits, including TPS, concurrency, and 24-hour scan windows, see PromQL limits and restrictions.

Sample request

The following `POST` request passes the PromQL expression in a form-encoded body. Special characters such as `{`, `}`, and `"` are percent-encoded.

```
POST /api/v1/query HTTP/1.1
Host: monitoring.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Authorization: AUTHPARAMS
X-Amz-Date: 20260605T193725Z
User-Agent: awscurl/0.36

query=sum(%7B%22http.server.active_requests%22%7D)
```

The same call with `awscurl`:

```
awscurl --service monitoring --region us-east-1 \
    -X POST 'https://monitoring.us-east-1.amazonaws.com/api/v1/query' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -d 'query=sum({"http.server.active_requests"})'
```

Sample response

A successful response uses the standard Prometheus JSON envelope:

```
HTTP/1.1 200 OK
x-amzn-RequestId: 12345678-abcd-4442-b8c5-262b45e9b535
Content-Type: application/json

{
    "status": "success",
    "data": {
        "resultType": "vector",
        "result": [
            {
                "metric": {},
                "value": [
                    1780000000.000,
                    "42"
                ]
            }
        ]
    }
}
```

When the result set exceeds the per-call cap, CloudWatch returns a truncated `result` array and includes a message in the `warnings` field:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "success",
    "data": {
        "resultType": "vector",
        "result": [ ... up to 500 series ... ]
    },
    "warnings": [
        "result truncated to the maximum of 500 series; refine the query or use the
 limit parameter"
    ]
}
```

GetSeries

The `GetSeries` operation returns the list of time series whose labels match one or more selectors.

Valid HTTP verbs

`GET, POST`

Valid URI

`/api/v1/series`

The full request URL combines the CloudWatch monitoring host for the AWS Region with the operation path, for example `https://monitoring.`*`AWS Region`*`.amazonaws.com/api/v1/series`. For information about endpoints, signing, and required IAM permissions, see PromQL querying.

URL query parameters

The following parameters are passed in the URL query string for `GET` requests, or as form-encoded body fields for `POST` requests.

Parameter	Description
`match[]`	Required. A series selector that filters the time series to return. Each selector must include an exact match (=) on the metric name — for

Parameter	Description
	example, {"http.server.active_requests"} or {__name__ ="http.server.active_requests"} . Selectors without a metric-name equal matcher are not supported. In addition to exact match, label matchers also support not equal (!=), regex match (=~), and negative regex match (!~). Specify match[] one or more times to combine selectors. For more information about selector syntax, see PromQL querying.
start	Required. Start of the time range to consider, as an RFC 3339 timestamp or a Unix timestamp.
end	Required. End of the time range to consider, as an RFC 3339 timestamp or a Unix timestamp.
limit	Optional. Maximum number of unique time series to return, from 1 to 10000. If you do not specify limit, CloudWatch returns up to the maximum (10,000). See Result limit.

Required IAM permissions

To call GetSeries, the calling identity must have the following IAM action:

- cloudwatch:ListMetrics

For the complete IAM action mapping for all PromQL operations, see IAM permissions for PromQL.

Result limit

A single /api/v1/series response can return up to **10,000 unique time series**. To request fewer results, pass the limit parameter with a value from 1 to 10000. If you do not specify limit, CloudWatch returns up to the maximum.

When the matching series exceed the cap, the response is truncated and a message is included in the standard Prometheus warnings field. The HTTP status code remains 200.

For the full list of PromQL limits, including TPS, concurrency, and 24-hour scan windows, see PromQL limits and restrictions.

Sample requests

The following POST request uses an exact match selector. The selector, `start`, and end are passed as form-encoded fields in the request body. Special characters such as {, }, ", and , are percent-encoded. The decoded `match[]` value is `{"http.server.active_requests","@resource.service.name"="myservice"}`.

```
POST /api/v1/series HTTP/1.1
Host: monitoring.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Authorization: AUTHPARAMS
X-Amz-Date: 20260605T193725Z
User-Agent: awscurl/0.36

match%5B%5D=%7B%22http.server.active_requests%22%2C%22%40resource.service.name%22%3D
%22myservice%22%7D&start=1780662000&end=1780665600
```

The following POST request uses a regex match selector. The decoded `match[]` value is `{"http.server.active_requests","@aws.region"=~"us-.*"}`.

```
POST /api/v1/series HTTP/1.1
Host: monitoring.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Authorization: AUTHPARAMS
X-Amz-Date: 20260605T193725Z
User-Agent: awscurl/0.36

match%5B%5D=%7B%22http.server.active_requests%22%2C%22%40aws.region%22%3D~%22us-.*
%22%7D&start=1780662000&end=1780665600
```

The same call with `awscurl`:

```
awscurl --service monitoring --region us-east-1 \
    -X POST 'https://monitoring.us-east-1.amazonaws.com/api/v1/series' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -d
 'match[]={"http.server.active_requests","@aws.region"=~"us-.*"}&start=1780662000&end=178066560
```

Sample response

A successful response uses the standard Prometheus JSON envelope. The `data` field is an array of label maps, one per matched series.

```
HTTP/1.1 200 OK
x-amzn-RequestId: 12345678-abcd-4442-b8c5-262b45e9b535
Content-Type: application/json

{
    "status": "success",
    "data": [
        {
            "__name__": "http.server.active_requests",
            "@resource.service.name": "myservice",
            "@aws.region": "us-east-1"
        },
        {
            "__name__": "http.server.active_requests",
            "@resource.service.name": "myservice",
            "@aws.region": "us-west-2"
        }
    ]
}
```

GetLabels

The `GetLabels` operation returns label names that are present in the metric store, or returns the values for a specific label name.

Valid HTTP verbs

- `GET`, `POST` for `/api/v1/labels`.
- `GET` for `/api/v1/label/`*`label_name`*`/values`.

Valid URIs

- `/api/v1/labels` — returns the list of label names.
- `/api/v1/label/`*`label_name`*`/values` — returns the list of values for the specified label.

The full request URL combines the CloudWatch monitoring host for the AWS Region with the operation path, for example `https://monitoring.AWS Region.amazonaws.com/api/v1/labels`. For information about endpoints, signing, and required IAM permissions, see PromQL querying.

URL query parameters

The following parameters are passed in the URL query string for `GET` requests, or as form-encoded body fields for `POST` requests.

Parameter	Applies to	Description
`match[]`	Both	Optional. A series selector that restricts which series are considered when computing the result. Selectors support exact match (=), not equal (!=), regex match (=~), and negative regex match (!~). Specify `match[]` one or more times to combine selectors. For more information about selector syntax, see PromQL querying.
`start`	Both	Optional. Start of the time range to consider, as an RFC 3339 timestamp or a Unix timestamp.
`end`	Both	Optional. End of the time range to consider, as an RFC 3339 timestamp or a Unix timestamp.
`limit`	Both	Optional. Maximum number of unique labels to return, from 1 to `10000`. If you do not specify `limit`, CloudWatch returns up to the maximum (10,000). See Result limit.

Required IAM permissions

To call `GetLabels` on either path, the calling identity must have the following IAM action:

- `cloudwatch:ListMetrics`

For the complete IAM action mapping for all PromQL operations, see IAM permissions for PromQL.

Result limit

A single `/api/v1/labels` or `/api/v1/label/label_name/values` response can return up to **10,000 labels**. To request fewer results, pass the `limit` parameter with a value from `1` to `10000`. If you do not specify `limit`, CloudWatch returns up to the maximum.

When the matching labels exceed the cap, the response is truncated and a message is included in the standard Prometheus `warnings` field. The HTTP status code remains `200`.

For the full list of PromQL limits, including TPS, concurrency, and 24-hour scan windows, see PromQL limits and restrictions.

Sample requests

List all label names with a POST request and an empty body:

```
POST /api/v1/labels HTTP/1.1
Host: monitoring.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Authorization: AUTHPARAMS
X-Amz-Date: 20260605T193725Z
User-Agent: awscurl/0.36
```

The same call with `awscurl`:

```
awscurl --service monitoring --region us-east-1 \
    -X POST 'https://monitoring.us-east-1.amazonaws.com/api/v1/labels' \
    -H 'Content-Type: application/x-www-form-urlencoded'
```

List label names matching a selector. The selector and time range are passed as form-encoded fields in the request body:

```
awscurl --service monitoring --region us-east-1 \
    -X POST 'https://monitoring.us-east-1.amazonaws.com/api/v1/labels' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -d
 'match[]={"http.server.active_requests","@aws.region"=~"us-.*"}&start=1780662000&end=178066560
```

List the values for the `@resource.service.name` label. The label name is part of the URL path and is percent-encoded (`@` → `%40`). This path supports only `GET`:

```
GET /api/v1/label/%40resource.service.name/values HTTP/1.1
Host: monitoring.us-east-1.amazonaws.com
Authorization: AUTHPARAMS
X-Amz-Date: 20260605T193725Z
User-Agent: awscurl/0.36
```

The same call with `awscurl`:

```
awscurl --service monitoring --region us-east-1 \
    'https://monitoring.us-east-1.amazonaws.com/api/v1/label/@resource.service.name/
values'
```

Sample responses

A successful response uses the standard Prometheus JSON envelope. The `data` field is an array of strings.

Response for `/api/v1/labels`:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "success",
    "data": [
        "__name__",
        "@aws.account",
        "@aws.region",
        "@resource.service.name",
        "@resource.cloud.region",
        "InstanceId"
    ]
}
```

Response for `/api/v1/label/`*`label_name`*`/values`:

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json

{
    "status": "success",
    "data": [
        "myservice",
        "checkout-api",
        "billing-worker"
    ]
}
```

Anomaly detection using PromQL

You can build anomaly detection bands for any Prometheus-compatible metric by using standard PromQL functions such as `quantile_over_time`, `stddev_over_time`, and `avg_over_time`. This approach computes a baseline and adds or subtracts a scaled standard deviation to define upper and lower bounds that adapt to your metric's natural patterns.

This works for any metric that returns a float value, such as CPU usage, request latency, or error counts. For information about ingesting metrics using OpenTelemetry, see OpenTelemetry.

Defining upper and lower bounds

To define an expected range for a metric, compute a baseline using the median or average over a time window, then add and subtract a multiple of the standard deviation. The multiplier controls sensitivity—higher values produce wider bands with fewer false positives, while lower values catch smaller deviations.

The following example creates an upper bound for an ad request metric using a 60-minute window and a multiplier of 3:

```
quantile_over_time(0.5, {"app.ads.ad_requests"}[60m] offset 1m)
  + 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m)
```

The following example creates the corresponding lower bound. The `clamp_min` function prevents the lower bound from going negative for metrics that can't have negative values:

```
clamp_min(
    quantile_over_time(0.5, {"app.ads.ad_requests"}[60m] offset 1m)
    - 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m),
```

```
0)
```

You can graph both bounds together in CloudWatch Query Studio to visualize the expected range for your metric. For more information, see Running PromQL queries in Query Studio.

Detecting breaches

To detect when a metric is outside its expected range, combine both bounds into a single query. The following expression returns only the data points where the metric value exceeds the upper bound or falls below the lower bound:

```
1 * {"app.ads.ad_requests"} > quantile_over_time(0.5, {"app.ads.ad_requests"}[60m]
 offset 1m)
    + 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m)
or
1 * {"app.ads.ad_requests"} < clamp_min(
    quantile_over_time(0.5, {"app.ads.ad_requests"}[60m] offset 1m)
    - 3 * stddev_over_time({"app.ads.ad_requests"}[60m] offset 1m),
0)
```

This query works across multiple label values, so you can detect anomalies across your entire fleet in a single query. You can use this expression to create a PromQL alarm that triggers when any time series breaches the expected range. For more information, see Creating a CloudWatch alarm using PromQL for anomaly detection.

CloudWatch detailed monitoring for OpenTelemetry metrics

Detailed monitoring gives you higher-resolution OpenTelemetry metrics for supported resources, which helps you diagnose performance issues faster. You can enable detailed monitoring for individual AWS resources through one centralized Amazon CloudWatch API, without building custom per-service infrastructure.

You identify each resource by its Amazon Resource Name (ARN). A resource can have only one configuration at a time.

Key concepts for ResourceMetricsConfiguration

Before you use ResourceMetricsConfiguration, review these terms. The terms explain how a configuration maps to a resource. They also explain which metrics CloudWatch collects by default and what happens when you update a configuration.

One configuration per resource

Each AWS resource identified by its ARN can have at most one ResourceMetricsConfiguration. The resource ARN serves as the unique identifier for the configuration. You do not need a separate configuration ID.

Metric selections

You can optionally specify which detailed metrics to collect by providing a `MetricSelections` list. Each selection contains an `IncludeMetrics` field with metric names. You can include between 1 and 500 metric names for each selection.

Default behavior

When you create a configuration without specifying `MetricSelections`, CloudWatch collects all available detailed metrics for the resource. When you specify `MetricSelections`, CloudWatch collects only the listed metrics.

Full replacement on update

When you update a configuration, the new `MetricSelections` value completely replaces the previous value. To add metrics to an existing selection, include the full list of metrics that you want to collect.

To start collecting detailed metrics for a resource, see Enabling detailed monitoring for a resource.

Enabling detailed monitoring for a resource

To enable detailed monitoring for a supported resource, create a ResourceMetricsConfiguration. You can use the AWS CLI or the `AWS::CloudWatch::ResourceMetricsConfiguration` CloudFormation resource type.

> **Note**
>
> If you attempt to create a ResourceMetricsConfiguration for a resource type that does not support detailed monitoring, the operation returns a `ValidationException`.

To enable detailed monitoring (AWS CLI)

1. Run the following command, replacing the placeholder ARN with the ARN of the resource that you want to monitor.

```
aws cloudwatch create-resource-metrics-configuration \
    --resource-arn arn:aws:service:us-east-1:123456789012:resource-type/resource-id
```

2. To collect only specific metrics instead of all available detailed metrics, include the `--metric-selections` parameter.

```
aws cloudwatch create-resource-metrics-configuration \
    --resource-arn arn:aws:service:us-east-1:123456789012:resource-type/resource-id
 \
    --metric-selections '[{"IncludeMetrics": ["MetricName1", "MetricName2"]}]'
```

To enable detailed monitoring (CloudFormation)

- Add the `AWS::CloudWatch::ResourceMetricsConfiguration` resource type to your CloudFormation template.

```
Resources:
  DetailedMonitoring:
    Type: AWS::CloudWatch::ResourceMetricsConfiguration
    Properties:
      ResourceArn: arn:aws:service:us-east-1:123456789012:resource-type/resource-id
```

Checking if detailed monitoring is enabled

To check whether detailed monitoring is enabled for a specific resource, use the `GetResourceMetricsConfiguration` operation.

To check detailed monitoring status (AWS CLI)

1. Run the following command for the resource that you want to check.

```
aws cloudwatch get-resource-metrics-configuration \
    --resource-arn arn:aws:service:us-east-1:123456789012:resource-type/resource-id
```

2. If detailed monitoring is enabled, the command returns the configuration details, including any metric selections. If no configuration exists, the command returns an error.

Updating the metric selection

You can update an existing ResourceMetricsConfiguration to change which metrics CloudWatch collects. The update fully replaces the `MetricSelections` value.

To update the metric selection (AWS CLI)

1. Run the following command. Include the complete list of metrics you want to collect.

```
aws cloudwatch update-resource-metrics-configuration \
    --resource-arn arn:aws:service:us-east-1:123456789012:resource-type/resource-id
 \
    --metric-selections '[{"IncludeMetrics": ["MetricName1", "MetricName2"]}]'
```

2. Verify the update by calling `GetResourceMetricsConfiguration`.

> **Important**
>
> The `--metric-selections` parameter fully replaces the previous value. To add a metric to an existing selection, include all previously selected metrics in addition to the new metric.

Disabling detailed monitoring

To disable detailed monitoring for a resource, delete its ResourceMetricsConfiguration. After you delete the configuration, CloudWatch stops collecting detailed metrics for that resource.

To disable detailed monitoring (AWS CLI)

1. Run the following command for the resource that you no longer want to monitor.

```
aws cloudwatch delete-resource-metrics-configuration \
    --resource-arn arn:aws:service:us-east-1:123456789012:resource-type/resource-id
```

2. Verify that the configuration was deleted by calling `GetResourceMetricsConfiguration`. The command returns an error if no configuration exists.

Required IAM permissions for ResourceMetricsConfiguration

Ensure your IAM policy includes the actions for the operations you use.

Operation	Required IAM action
Create a configuration	`cloudwatch:CreateResourceMetricsConfiguration`
Get a configuration	`cloudwatch:GetResourceMetricsConfiguration`
Update a configuration	`cloudwatch:UpdateResourceMetricsConfiguration`
Delete a configuration	`cloudwatch:DeleteResourceMetricsConfiguration`

The following example IAM policy grants permissions for all ResourceMetricsConfiguration operations, scoped to a single resource with the `cloudwatch:ResourceArn` condition key.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:CreateResourceMetricsConfiguration",
                "cloudwatch:GetResourceMetricsConfiguration",
                "cloudwatch:UpdateResourceMetricsConfiguration",
                "cloudwatch:DeleteResourceMetricsConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:ResourceArn": "arn:aws:service:us-
east-1:123456789012:resource-type/resource-id"
                }
            }
        }
```

```
    ]
}
```

> **Note**
>
> The ResourceMetricsConfiguration APIs do not support resource-level permissions. You must use `"Resource": "*"` in your IAM policy and scope access with the `cloudwatch:ResourceArn` condition key, as shown in the preceding example.

For more information about `cloudwatch:ResourceArn`, including how to match multiple resources with wildcards, see Condition keys for resource metrics configuration access.

OTel metrics pricing and storage

OpenTelemetry metrics in CloudWatch use a per-GB ingestion model with included storage and query access. There are no per-metric, per-API-call, or separate storage charges. For current pricing details, see Amazon CloudWatch Pricing.

Key pricing characteristics

The OTel metrics pricing model has the following characteristics:

- **Per-GB ingestion** — You pay based on the volume of data ingested, not the number of unique metric series.
- **Storage included** — 15 months retention at no additional charge.
- **Console queries free** — PromQL queries in the CloudWatch console (Query Studio, dashboards) are free.
- **API queries charged per sample** — Programmatic PromQL queries are charged per million samples scanned.
- **No per-metric charge** — Unlike Classic pricing, there is no cost per unique metric name or label combination.

How OTel compares to Classic pricing

The following table compares OTel metrics pricing with Classic metrics pricing.

Feature	OTel Metrics	Classic Metrics
Billing unit	Per GB ingested	Per metric per month
Storage	15 months included	15 months included
Console queries	Free	Free
API queries	Per million samples scanned	Per 1,000 metrics requested (GetMetricData)
High cardinality cost	Linear with data volume (not metric count)	Expensive — each unique dimension combination is a separate billable metric

What counts as ingested data

Each OTLP data point contributes its serialized size to the ingestion volume. The serialized size includes the following components:

- Metric name
- All labels (keys and values)
- Resource attributes
- Timestamp and value

Larger label sets produce more bytes per data point. To see the cost that your OTel metric ingestion generates, filter your cost and usage data by `usage type` in Cost Explorer. For more information about CloudWatch usage types, see To visualize and analyze CloudWatch cost and usage data. For the rates that apply to each usage type, see Amazon CloudWatch Pricing.

Cost optimization tips

Use the following strategies to optimize your OTel metrics costs:

- Use shorter label values where possible. Label values contribute to the byte size of each data point.
- Drop high-cardinality labels that you don't query (for example, request IDs) at the collector level.

- For Container Insights OTel, the default collection generates a predictable volume based on cluster size.

Migrate from Classic to OTel metrics

Customers currently publishing custom metrics through PutMetricData or EMF can migrate incrementally to the OTel path. There is no flag day — migration happens workload by workload at your own pace.

Migration approach

The migration follows four phases:

1. **Dual-write** — Publish metrics through both Classic and OTel paths simultaneously.
2. **Validate** — Confirm OTel metrics appear in Query Studio through PromQL.
3. **Recreate consumers** — Update alarms and dashboards to use PromQL and OTel metric names.
4. **Cut over** — Stop Classic publishing after you validate OTel consumers.

Step 1: Instrument with OTel SDK (dual-write)

Add the OTel SDK alongside your existing PutMetricData calls. Both paths can publish simultaneously — you don't lose data.

```
# Existing Classic publishing (keep running during migration)
cloudwatch.put_metric_data(
    Namespace='MyApp',
    MetricData=[{'MetricName': 'RequestLatency', 'Value': 42.5, 'Unit':
 'Milliseconds'}]
)

# New OTel publishing (add this)
from opentelemetry import metrics
meter = metrics.get_meter("my-app")
histogram = meter.create_histogram("http_request_duration_seconds")
histogram.record(0.0425, {"method": "GET", "path": "/api/users"})
```

Step 2: Verify in Query Studio

To verify that your OTel metrics are arriving, open the CloudWatch console and navigate to Query Studio. Search for your metric:

```
http_request_duration_seconds
```

Confirm that data appears with the labels you expect.

Step 3: Recreate alarms on OTel metrics

Create new alarms that query OTel metrics using PromQL expressions. The following example shows a Classic alarm and its OTel equivalent.

Classic alarm:

```
aws cloudwatch put-metric-alarm \
  --alarm-name "high-latency" \
  --namespace "MyApp" \
  --metric-name "RequestLatency" \
  --statistic Average --threshold 100 ...
```

OTel equivalent (PromQL alarm):

```
aws cloudwatch put-metric-alarm \
  --alarm-name "high-latency-otel" \
  --metrics '[{"Id":"q1","Expression":"avg(http_request_duration_seconds{path=\"/api/
users\"}) * 1000","Period":300,"ReturnData":true}]' \
  --threshold 100 ...
```

Run both alarms in parallel until you are confident the OTel version fires correctly.

Step 4: Stop Classic publishing

After you validate your OTel alarms and dashboards, remove the PutMetricData calls from your application code. Classic metrics stop incurring charges immediately.

Metric name mapping

The following table shows common Classic metric names and their suggested OTel equivalents.

Classic name	Suggested OTel name	Notes
RequestLatency (ms)	http_request_duration_seconds	Convert to seconds (OTel convention)
RequestCount	http_requests_total	Use `_total` suffix for counters
ErrorCount	http_server_errors_total	Use `_total` suffix
QueueDepth	queue_depth	Gauge — no suffix needed

What about AWS vended metrics?

You don't need to migrate AWS vended metrics (Amazon EC2 CPU, Amazon RDS connections, and so on) manually. Enable OTel Vended Metric Enrichment to make them queryable through PromQL automatically. For more information, see AWS vended metrics in OpenTelemetry format.

AWS vended metrics in OpenTelemetry format

> **Tip**
>
> To learn more about OpenTelemetry on CloudWatch, check out the Cloud Operations Enablement workshop and event series on the AWS Experience website.

You can enable OTel enrichment to make vended metrics queryable through PromQL. For the resources and metrics that support this, see Supported AWS resources and metrics for OpenTelemetry enrichment. After you enable OTel enrichment, metrics that contain a resource identifier dimension (for example, EC2 CPUUtilization with an InstanceId dimension) are enriched with resource ARN and resource tag labels and become queryable using PromQL.

The enriched metric preserves the original metric name and CloudWatch dimensions, and adds:

- **Resource attributes** – the resource ARN (`cloud.resource_id`), cloud provider, region, and account ID.
- **Instrumentation scope** – identifies the source service and marks the metric as OTel-enriched.
- **Resource tags** – any AWS resource tags associated with the resource, queryable as PromQL labels.

The original classic CloudWatch metric is not modified and remains available through existing CloudWatch APIs.

Enabling OpenTelemetry enrichment for vended metrics

Before you start OTel enrichment, you must enable resource tags on telemetry for your account.

You can enable OTel enrichment for your account in a specific region using the CloudWatch console, AWS CLI, CloudFormation, Terraform, or AWS SDK.

You will need permissions for the following operation: `cloudwatch:StartOTelEnrichment`

To enable OTel enrichment for AWS metrics (CloudWatch Console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **CloudWatch**, then choose **Settings**.
3. In the **Enable OTel Enrichment for AWS Metrics** pane, toggle the feature On.

AWS CLI

```
aws cloudwatch start-otel-enrichment
```

CloudFormation

> **Note**
>
> The `AWS::CloudWatch::OTelEnrichment` resource requires the `AWS::ObservabilityAdmin::TelemetryEnrichment` resource to be configured first. For more information, see enable resource tags on telemetry.

```
Resources:
  TelemetryEnrichmentExample:
    Type: AWS::ObservabilityAdmin::TelemetryEnrichment
  OTelEnrichmentExample:
    Type: AWS::CloudWatch::OTelEnrichment
    DependsOn: TelemetryEnrichmentExample
```

Terraform

> **Note**
>
> The `aws_cloudwatch_otel_enrichment` Terraform resource requires the `aws_observabilityadmin_telemetry_enrichment` resource to be configured first.

```
resource "aws_observabilityadmin_telemetry_enrichment" "example" {
}

resource "aws_cloudwatch_otel_enrichment" "example" {
  depends_on = [aws_observabilityadmin_telemetry_enrichment.example]
}
```

For more information, see aws_cloudwatch_otel_enrichment in the Terraform Registry.

To enable across multiple regions, create the same resource in each regional stack or invoke the API in each region of interest.

After enrichment is enabled, you can start querying vended metrics through PromQL. See: Querying vended AWS metrics with PromQL.

Filtering enrichment by namespace and metric

CloudWatch OTel enrichment decorates your account's vended metrics with resource ARN and resource tag labels. You can then group and filter by resource attributes in PromQL and Query Studio. By default, enrichment applies to every namespace that CloudWatch supports for enrichment. You can narrow enrichment to a chosen subset of namespaces and metric names by using two optional parameters, `IncludeFilters` and `ExcludeFilters`. You can also change those filters on a running account with the `UpdateOTelEnrichment` operation. Use filtering to optimize enrichment cost when you need only some of your metrics in OTel.

Before you filter enrichment, you must start enrichment for your account. For more information, see Enabling OpenTelemetry enrichment for vended metrics.

Related

- Enabling OpenTelemetry enrichment for vended metrics
- Querying vended AWS metrics with PromQL
- Enabling resource tags on telemetry

Operations that set enrichment filters

You set enrichment filters with two operations. Each operation applies to your account and is scoped to a single AWS Region.

- `StartOTelEnrichment` — Starts enrichment, and optionally sets the initial filters.
- `UpdateOTelEnrichment` — Replaces the filters of an account that is already running enrichment.

The metric selector

A metric selector is an `OTelEnrichmentMetricSelector` data type that picks the metrics in exactly one namespace. Both `IncludeFilters` and `ExcludeFilters` are lists of these selectors. A selector has the following members.

`Namespace`

Type: String. Length: 1–255 characters. This member is required, and it must not start with a colon. CloudWatch matches the namespace exactly and case-sensitively.

`MetricNames`

Type: list of String. Each name is 1–255 characters, and the list holds 0–100 items. This member is optional. Omit it to match every metric in the namespace.

How filters are evaluated

CloudWatch evaluates `IncludeFilters` and `ExcludeFilters` according to the following rules.

- **Include first, then exclude** – CloudWatch applies `IncludeFilters` first, then subtracts the `ExcludeFilters` match set from the result. Exclude always takes precedence, so a metric matched by both filters is not enriched.
- **Empty and absent are equivalent** – An omitted list, a null list, and an empty list (`[]`) mean the same thing everywhere, including `MetricNames` within a selector.
- **Exact, case-sensitive matching** – CloudWatch matches names exactly and case-sensitively, with no wildcards, prefixes, or normalization. `AWS/EC2` matches only `AWS/EC2`, and `aws/ec2` matches nothing.
- **An inactive direction is permissive** – When you set no `IncludeFilters`, every supported namespace is in scope. When you set no `ExcludeFilters`, nothing is removed.

The following table shows what gets enriched for each combination of filters.

`IncludeFilters`	`ExcludeFilters`	What gets enriched
Absent or empty	Absent or empty	Every namespace that CloudWatch supports for enrichment.
Present	Absent or empty	Only what `IncludeFilters` matches.
Absent or empty	Present	Everything supported, minus what `ExcludeFilters` matches.
Present	Present	The `IncludeFilters` match set, minus the `ExcludeFilters` match set.

Example: Enrich a subset of metrics

Suppose that you want to enrich all of `AWS/EC2` except two network metrics, plus exactly two `AWS/RDS` metrics, and nothing else. The following filter configuration produces that result.

```
{
  "IncludeFilters": [
    { "Namespace": "AWS/EC2" },
    { "Namespace": "AWS/RDS", "MetricNames": ["CPUUtilization",
 "DatabaseConnections"] }
  ],
  "ExcludeFilters": [
    { "Namespace": "AWS/EC2", "MetricNames": ["NetworkPacketsIn",
 "NetworkPacketsOut"] }
  ]
}
```

This configuration produces the following result.

- Every `AWS/EC2` metric except `NetworkPacketsIn` and `NetworkPacketsOut` is enriched.
- `AWS/RDS` `CPUUtilization` and `DatabaseConnections` are enriched.
- Every other `AWS/RDS` metric is not enriched, because `IncludeFilters` did not name it.
- Every other namespace is not enriched, because `IncludeFilters` is active and does not list it.

This configuration consumes 3 of the 100 available selectors: two include selectors and one exclude selector.

Starting enrichment on an account that is already running

Calling `StartOTelEnrichment` on an account that is already running enrichment is a no-op. It does not return an error, and it does not apply the filters in the request. The stored configuration is unchanged.

The response makes this behavior observable. `StartOTelEnrichment` returns the filters that were already stored, not the filters that the request carried. If you send filters and get different filters back, enrichment was already running, and the filters you sent were discarded.

To change filters on a running account, use `UpdateOTelEnrichment`. Do not stop and restart enrichment to change filters, because a restart resets `CreatedAt` and creates an enrichment gap.

Updating filters

`UpdateOTelEnrichment` replaces filters. It does not merge them. `IncludeFilters` and `ExcludeFilters` move as a pair, so whatever the request omits is cleared. The following table shows the effect of each request shape on the stored configuration.

Request	Effect on stored configuration
Both specified	Both are replaced.
Only `IncludeFilters`	`IncludeFilters` is replaced, and `ExcludeFilters` is cleared.
Only `ExcludeFilters`	`ExcludeFilters` is replaced, and `IncludeFilters` is cleared.
Neither specified	Both are cleared, and enrichment reverts to every supported namespace.

The last write wins. There is no read-modify-write helper, so to add one selector you send the full intended list. If you do not already track the current state, read it with `GetOTelEnrichment` first.

`UpdateOTelEnrichment` requires enrichment to already be running. Against a stopped account, it returns `ResourceNotFoundException` (HTTP 404), so start enrichment first.

Timestamps

`GetOTelEnrichment` returns `CreatedAt` and `UpdatedAt`.

- After `StartOTelEnrichment` starts enrichment on an account that isn't already running it, `CreatedAt` is set to the current time, and `UpdatedAt` equals `CreatedAt`. On an account that is already running enrichment, `StartOTelEnrichment` is a no-op and leaves both timestamps unchanged.
- After `UpdateOTelEnrichment`, `CreatedAt` is unchanged, and `UpdatedAt` is set to the current time.

An `UpdatedAt` value that is greater than `CreatedAt` indicates that the filters have changed at least once since enrichment started.

Reading enrichment state

`GetOTelEnrichment` returns `Status` (`Running` or `Stopped`) plus the filters and timestamps. Every member below `Status` is omitted when enrichment is stopped.

When enrichment is running, an omitted `IncludeFilters` is not unknown. It means that no include direction is stored, so every supported namespace is in scope. Likewise, an omitted `ExcludeFilters` means that nothing is excluded. Treat an absent filter as the permissive case, never as an error.

Limits

The following table lists the limits for enrichment filters.

Limit	Value	Enforced by
Selectors across `IncludeFilters` and `ExcludeFilters` combined	100	Service (the binding limit)
Selectors per individual list	100	Model length backstop
Metric names per selector	100	Model length
Namespace length	1–255 characters, must not begin with a colon	Model

Limit	Value	Enforced by
`MetricName` length	1–255 characters	Model

The combined cap is the limit that rejects requests. Two lists of 60 selectors each satisfy the per-list limit of 100, but they sum to 120 and fail with `ValidationException`.

Errors

The following table lists the errors that each operation can return.

Operation	Errors
`StartOTelEnrichment`	`ValidationException`
`UpdateOTelEnrichment`	`ValidationException` , `ResourceNotFoundException`
`GetOTelEnrichment`	None
`StopOTelEnrichment`	None

`ValidationException` covers malformed namespaces, over-long names, and exceeding the combined-selector cap. `ResourceNotFoundException` on `UpdateOTelEnrichment` means exactly one thing: enrichment is not running for this account. The framework raises throttling, internal, and access-denied errors. They are not modeled on these operations, and you handle them as you do for any other CloudWatch call.

Managing filters with the AWS CLI

To start enrichment with initial filters, run the following command.

```
aws cloudwatch start-otel-enrichment \
    --include-filters '[{"Namespace":"AWS/EC2"},{"Namespace":"AWS/RDS","MetricNames":
["CPUUtilization"]}]' \
    --exclude-filters '[{"Namespace":"AWS/EC2","MetricNames":["NetworkPacketsIn"]}]'
```

To read the current state, run the following command.

```
aws cloudwatch get-otel-enrichment
```

To replace the filters, run the following command. This command clears anything that you omit.

```
aws cloudwatch update-otel-enrichment \
    --include-filters '[{"Namespace":"AWS/EC2"},{"Namespace":"AWS/Lambda"}]' \
    --exclude-filters '[]'
```

To revert to enriching every supported namespace, run the following command.

```
aws cloudwatch update-otel-enrichment
```

To stop enrichment, run the following command.

```
aws cloudwatch stop-otel-enrichment
```

To verify that a change took effect, read it back and compare `UpdatedAt`.

```
aws cloudwatch get-otel-enrichment --query
 '{S:Status,U:UpdatedAt,I:IncludeFilters,E:ExcludeFilters}'
```

Best practices

We recommend the following practices when you filter enrichment.

- Start enrichment once, then use `UpdateOTelEnrichment` to change filters. Never restart to change filters.
- Send the complete intended filter set on every update, because omission clears.
- Copy namespace names exactly from `ListMetrics` output. Matching is case-sensitive with no wildcards.
- Budget selectors against the combined limit of 100, not 100 per list.
- Read absent filters as permissive, not as missing data.
- Confirm writes from the response rather than the request that you sent. On the `StartOTelEnrichment` no-op path they differ, and that difference is the signal.

Supported AWS resources and metrics for OpenTelemetry enrichment

When you enable OTel enrichment, CloudWatch republishes selected metrics from the following AWS services as OpenTelemetry time series. Each CloudWatch namespace maps to an OpenTelemetry instrumentation scope named `cloudwatch.aws/`*`service`* — for example, the `AWS/EC2` namespace maps to the scope `cloudwatch.aws/ec2`. All enriched instruments use Delta temporality.

Depending on the resource, enriched metrics can also carry the following resource context:

- `cloud.resource_id` — the resource ARN, added when the metric can be resolved to a specific resource.
- `tag.*` — the resource's AWS tags, added when the resource is supported for the section called "Using resource tags for telemetry".

The following tables list, per namespace, the supported resources and their enriched metrics. For each metric, the OpenTelemetry metric name matches the CloudWatch metric name unless a rewritten name is shown, and the instrument type is either `Sum` or `Histogram`. The metric's CloudWatch dimensions are emitted verbatim as OpenTelemetry data point attributes (the attribute key is the dimension name); the *Data point attributes* column lists the dimension sets each metric is published with.

Topics

- AWS/AIDevOps
- AWS/AOSS
- AWS/ApiGateway
- AWS/AppFlow
- AWS/AppStream
- AWS/AppSync
- AWS/ApplicationELB
- AWS/Athena
- AWS/AutoScaling
- AWS/Backup
- AWS/Bedrock
- AWS/BedrockMantle

- AWS/ClientVPN
- AWS/CloudFront
- AWS/CloudWatch/MetricStreams
- AWS/CodeGuruProfiler
- AWS/Cognito
- AWS/Connect
- AWS/DAX
- AWS/DX
- AWS/DataSync
- AWS/DocDB
- AWS/DocDB-Elastic
- AWS/DynamoDB
- AWS/EBS
- AWS/EC2
- AWS/EC2CapacityReservations
- AWS/EFS
- AWS/EKS
- AWS/ELB
- AWS/EMRServerless
- AWS/ES
- AWS/ElastiCache
- AWS/ElasticBeanstalk
- AWS/ElasticMapReduce
- AWS/EventBridge/Pipes
- AWS/Events
- AWS/FSx
- AWS/Firehose
- AWS/FraudDetector
- AWS/GameLift
- AWS/GatewayELB

- AWS/IVSChat
- AWS/IoT
- AWS/KMS
- AWS/Kafka
- AWS/Kendra
- AWS/Kinesis
- AWS/KinesisAnalytics
- AWS/Lambda
- AWS/M2
- AWS/MWAA
- AWS/MediaTailor
- AWS/MemoryDB
- AWS/NATGateway
- AWS/NetworkELB
- AWS/NetworkFirewall
- AWS/Pinpoint
- AWS/Prometheus
- AWS/RDS
- AWS/RUM
- AWS/Redshift
- AWS/Redshift-Serverless
- AWS/Route53
- AWS/Route53Resolver
- AWS/S3
- AWS/SNS
- AWS/SQS
- AWS/SWF
- AWS/SageMaker
- AWS/Scheduler
- AWS/Transfer

- AWS/TransitGateway
- AWS/VPN
- AWS/VpcLattice
- AWS/WorkSpaces

AWS/AIDevOps

The `AWS/AIDevOps` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/aidevops`. All enriched instruments use Delta temporality.

- AWS::DevOpsAgent::AgentSpace

AWS::DevOpsAgent::AgentSpace

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConsumedChatRequests	ConsumedChatRequests	Sum	AgentSpaceUUID
ConsumedEvaluationTime	ConsumedEvaluationTime	Histogram	AgentSpaceUUID
ConsumedInvestigationTime	ConsumedInvestigationTime	Histogram	AgentSpaceUUID
ConsumedOnDemandTime	ConsumedOnDemandTime	Histogram	AgentSpaceUUID
ConsumedReleaseAPITestingTime	ConsumedReleaseAPITestingTime	Histogram	AgentSpaceUUID

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConsumedReleaseReadinessReviewTime	ConsumedReleaseReadinessReviewTime	Histogram	AgentSpaceUUID
ConsumedReleaseShepherdTime	ConsumedReleaseShepherdTime	Histogram	AgentSpaceUUID
ConsumedReleaseUITestingTime	ConsumedReleaseUITestingTime	Histogram	AgentSpaceUUID
TopologyCompletionCount	TopologyCompletionCount	Sum	AgentSpaceUUID

AWS/AOSS

The AWS/AOSS namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/aoss`. All enriched instruments use Delta temporality.

- AWS::OpenSearchServerless::Collection
- AWS::OpenSearchServerless::CollectionGroup

AWS::OpenSearchServerless::Collection

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
2xx	2xx	Sum	ClientId, CollectionId , CollectionName
3xx	3xx	Sum	ClientId, CollectionId , CollectionName
4xx	4xx	Sum	ClientId, CollectionId , CollectionName
5xx	5xx	Sum	ClientId, CollectionId , CollectionName
ActiveCollection	ActiveCollection	Histogram	ClientId, CollectionId , CollectionName
ActiveCollection-Shadow	ActiveCollection-Shadow	Histogram	ClientId, CollectionId , CollectionName
AutoOptimizeJob	AutoOptimizeJob	Sum	ClientId, CollectionId , CollectionName
IngestionDataRate-Shadow	IngestionDataRate-Shadow	Sum	ClientId, CollectionId , CollectionName
IngestionDocumentErrors-Shadow	IngestionDocumentErrors-Shadow	Sum	ClientId, CollectionId , CollectionName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Ingestion DocumentRate-Shadow	Ingestion DocumentRate-Shadow	Sum	ClientId, CollectionId , CollectionName
Ingestion RequestErrors	Ingestion RequestErrors	Sum	ClientId, CollectionId , CollectionName
Ingestion RequestErrors-Shadow	Ingestion RequestErrors-Shadow	Sum	ClientId, CollectionId , CollectionName
Ingestion RequestLatency	Ingestion RequestLatency	Histogram	ClientId, CollectionId , CollectionName
Ingestion RequestRate	Ingestion RequestRate	Histogram	ClientId, CollectionId , CollectionName
Ingestion RequestRate-Shadow	Ingestion RequestRate-Shadow	Histogram	ClientId, CollectionId , CollectionName
Ingestion RequestSuccess	Ingestion RequestSuccess	Sum	ClientId, CollectionId , CollectionName
Ingestion RequestSuccess-Shadow	Ingestion RequestSuccess-Shadow	Sum	ClientId, CollectionId , CollectionName
KNNRemote IndexBuildFailureCount	KNNRemote IndexBuildFailureCount	Sum	ClientId, CollectionId , CollectionName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KNNRemote IndexBuil dSuccessCount	KNNRemote IndexBuil dSuccessCount	Sum	ClientId, CollectionId , CollectionName
KNNRemote IndexBuildTime	KNNRemote IndexBuildTime	Histogram	ClientId, CollectionId , CollectionName
SearchReq uestErrors- Shadow	SearchReq uestErrors- Shadow	Sum	ClientId, CollectionId , CollectionName
SearchReq uestLatency	SearchReq uestLatency	Histogram	ClientId, CollectionId , CollectionName
SearchReq uestRate	SearchReq uestRate	Histogram	ClientId, CollectionId , CollectionName
SearchReq uestRate- Shadow	SearchReq uestRate- Shadow	Histogram	ClientId, CollectionId , CollectionName
SemanticS earchOCU	SemanticS earchOCU	Histogram	ClientId, CollectionId , CollectionName

AWS::OpenSearchServerless::CollectionGroup

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
IndexingOCU	IndexingOCU	Histogram	ClientId, CollectionGroupId , CollectionGroupName
SearchOCU	SearchOCU	Histogram	ClientId, CollectionGroupId , CollectionGroupName

AWS/ApiGateway

The AWS/ApiGateway namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/apigateway. All enriched instruments use Delta temporality.

- AWS::ApiGateway::Stage
- stage

AWS::ApiGateway::Stage

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
4XXError	4XXError	Sum	ApiName, Stage
4XXError	4XXErrorByMethod	Sum	ApiName, Method, Resource, Stage

Metric	OpenTelemetry metric	Instrument type	Data point attributes
5XXError	5XXError	Sum	ApiName, Stage
5XXError	5XXErrorByMethod	Sum	ApiName, Method, Resource, Stage
CacheHitCount	CacheHitCount	Sum	ApiName, Stage
CacheHitCount	CacheHitCountByMethod	Sum	ApiName, Method, Resource, Stage
CacheMissCount	CacheMissCount	Sum	ApiName, Stage
CacheMissCount	CacheMissCountByMethod	Sum	ApiName, Method, Resource, Stage
Count	Count	Sum	ApiName, Stage
Count	CountByMethod	Sum	ApiName, Method, Resource, Stage
IntegrationLatency	IntegrationLatency	Histogram	ApiName, Stage
IntegrationLatency	IntegrationLatencyByMethod	Histogram	ApiName, Method, Resource, Stage
Latency	Latency	Histogram	ApiName, Stage
Latency	LatencyByMethod	Histogram	ApiName, Method, Resource, Stage

stage

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
4xx	4xx	Sum	ApiId, Stage
5xx	5xx	Sum	ApiId, Stage
ClientError	ClientError	Sum	ApiId, Stage
ConnectCount	ConnectCount	Sum	ApiId, Stage
Count	Count	Sum	ApiId, Stage
DataProcessed	DataProcessed	Histogram	ApiId, Stage
ExecutionError	ExecutionError	Sum	ApiId, Stage
IntegrationError	IntegrationError	Sum	ApiId, Stage
IntegrationLatency	IntegrationLatency	Histogram	ApiId, Stage
Latency	Latency	Histogram	ApiId, Stage
MessageCount	MessageCount	Sum	ApiId, Stage

AWS/AppFlow

The AWS/AppFlow namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/appflow. All enriched instruments use Delta temporality.

- AWS::AppFlow::Flow

AWS::AppFlow::Flow

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FlowExecutionRecordsProcessed	FlowExecutionRecordsProcessed	Sum	FlowName
FlowExecutionTime	FlowExecutionTime	Histogram	FlowName
FlowExecutionsFailed	FlowExecutionsFailed	Sum	FlowName
FlowExecutionsStarted	FlowExecutionsStarted	Sum	FlowName
FlowExecutionsSucceeded	FlowExecutionsSucceeded	Sum	FlowName

AWS/AppStream

The AWS/AppStream namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/appstream. All enriched instruments use Delta temporality.

- fleet

fleet

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveUserSessionCapacity	ActiveUserSessionCapacity	Histogram	Fleet
ActualCapacity	ActualCapacity	Histogram	Fleet
ActualUserSessionCapacity	ActualUserSessionCapacity	Histogram	Fleet
AvailableCapacity	AvailableCapacity	Histogram	Fleet
AvailableUserSessionCapacity	AvailableUserSessionCapacity	Histogram	Fleet
CapacityUtilization	CapacityUtilization	Histogram	Fleet
DesiredCapacity	DesiredCapacity	Histogram	Fleet
DesiredUserSessionCapacity	DesiredUserSessionCapacity	Histogram	Fleet
InUseCapacity	InUseCapacity	Histogram	Fleet
InsufficientCapacityError	InsufficientCapacityError	Sum	Fleet
InsufficientConcurrencyLimitError	InsufficientConcurrencyLimitError	Sum	Fleet

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PendingCapacity	PendingCapacity	Histogram	Fleet
PendingUserSessionCapacity	PendingUserSessionCapacity	Histogram	Fleet
ReservedCapacity	ReservedCapacity	Histogram	Fleet
RunningCapacity	RunningCapacity	Histogram	Fleet
RunningUserSessionCapacity	RunningUserSessionCapacity	Histogram	Fleet

AWS/AppSync

The AWS/AppSync namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/appsync. All enriched instruments use Delta temporality.

- AWS::AppSync::GraphQLApi

AWS::AppSync::GraphQLApi

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
4XXError	4XXError	Sum	GraphQLAPIId
5XXError	5XXError	Sum	GraphQLAPIId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveConnections	ActiveConnections	Histogram	GraphQLAPIId
ActiveSubscriptions	ActiveSubscriptions	Histogram	GraphQLAPIId
CacheHit	CacheHit	Sum	GraphQLAPIId , Resolver
CacheMiss	CacheMiss	Sum	GraphQLAPIId , Resolver
ConnectClientError	ConnectClientError	Sum	GraphQLAPIId
ConnectRequests	ConnectRequests	Sum	GraphQLAPIId
ConnectServerError	ConnectServerError	Sum	GraphQLAPIId
ConnectSuccess	ConnectSuccess	Sum	GraphQLAPIId
ConnectionDuration	ConnectionDuration	Histogram	GraphQLAPIId
DisconnectClientError	DisconnectClientError	Sum	GraphQLAPIId
DisconnectServerError	DisconnectServerError	Sum	GraphQLAPIId
DisconnectSuccess	DisconnectSuccess	Sum	GraphQLAPIId
InboundMessageDelayed	InboundMessageDelayed	Sum	GraphQLAPIId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InboundMessageDropped	InboundMessageDropped	Sum	GraphQLAPIId
InboundMessageError	InboundMessageError	Sum	GraphQLAPIId
InboundMessageFailure	InboundMessageFailure	Sum	GraphQLAPIId
InboundMessageSuccess	InboundMessageSuccess	Sum	GraphQLAPIId
InvalidationRequestDropped	InvalidationRequestDropped	Sum	GraphQLAPIId
InvalidationRequestError	InvalidationRequestError	Sum	GraphQLAPIId
InvalidationRequestFailure	InvalidationRequestFailure	Sum	GraphQLAPIId
InvalidationRequestSuccess	InvalidationRequestSuccess	Sum	GraphQLAPIId
InvalidationSuccess	InvalidationSuccess	Sum	GraphQLAPIId
OutboundMessages	OutboundMessages	Sum	GraphQLAPIId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PublishDataMessageClientError	PublishDataMessageClientError	Sum	GraphQLAPIId
PublishDataMessageServerError	PublishDataMessageServerError	Sum	GraphQLAPIId
PublishDataMessageSize	PublishDataMessageSize	Histogram	GraphQLAPIId
PublishDataMessageSuccess	PublishDataMessageSuccess	Sum	GraphQLAPIId
SubscribeClientError	SubscribeClientError	Sum	GraphQLAPIId
SubscribeServerError	SubscribeServerError	Sum	GraphQLAPIId
SubscribeSuccess	SubscribeSuccess	Sum	GraphQLAPIId
UnsubscribeClientError	UnsubscribeClientError	Sum	GraphQLAPIId
UnsubscribeServerError	UnsubscribeServerError	Sum	GraphQLAPIId
UnsubscribeSuccess	UnsubscribeSuccess	Sum	GraphQLAPIId

AWS/ApplicationELB

The `AWS/ApplicationELB` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/elasticloadbalancing`. All enriched instruments use Delta temporality.

- AWS::ElasticLoadBalancingV2::LoadBalancer
- AWS::ElasticLoadBalancingV2::TargetGroup

AWS::ElasticLoadBalancingV2::LoadBalancer

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveConnectionCount	ActiveConnectionCount	Histogram	AvailabilityZone , LoadBalancer
ActiveZonalShiftHostCount	ActiveZonalShiftHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
AnomalousHostCount	AnomalousHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
AppCookieNonStickinessCount	AppCookieNonStickinessCount	Sum	LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ClientTLSNegotiationErrorCount	ClientTLSNegotiationErrorCount	Sum	AvailabilityZone, LoadBalancer
ConsumedLCUs	ConsumedLCUs	Histogram	LoadBalancer
DesyncMitigationMode_NonCompliant_Request_Count	DesyncMitigationMode_NonCompliant_Request_Count	Sum	AvailabilityZone, LoadBalancer
DroppedInvalidHeaderRequestCount	DroppedInvalidHeaderRequestCount	Sum	AvailabilityZone, LoadBalancer
ELBAuthError	ELBAuthError	Sum	AvailabilityZone, LoadBalancer
ELBAuthFailure	ELBAuthFailure	Sum	AvailabilityZone, LoadBalancer
ELBAuthLatency	ELBAuthLatency	Histogram	AvailabilityZone, LoadBalancer
ELBAuthRefreshTokenSuccess	ELBAuthRefreshTokenSuccess	Sum	AvailabilityZone, LoadBalancer
ELBAuthSuccess	ELBAuthSuccess	Sum	AvailabilityZone, LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ELBAuthUserClaimsSizeExceeded	ELBAuthUserClaimsSizeExceeded	Sum	AvailabilityZone , LoadBalancer
ExcessiveLowReputationPackets	ExcessiveLowReputationPackets	Sum	LoadBalancer
ForwardedInvalidHeaderRequestCount	ForwardedInvalidHeaderRequestCount	Sum	AvailabilityZone , LoadBalancer
GrpcRequestCount	GrpcRequestCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup
HTTPCode_ELB_3XX_Count	HTTPCode_ELB_3XX_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_ELB_4XX_Count	HTTPCode_ELB_4XX_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_ELB_500_Count	HTTPCode_ELB_500_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_ELB_502_Count	HTTPCode_ELB_502_Count	Sum	AvailabilityZone , LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HTTPCode_ELB_503_Count	HTTPCode_ELB_503_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_ELB_504_Count	HTTPCode_ELB_504_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_ELB_5XX_Count	HTTPCode_ELB_5XX_Count	Sum	AvailabilityZone , LoadBalancer
HTTPCode_Target_2XX_Count	HTTPCode_Target_2XX_Count	Sum	AvailabilityZone , LoadBalancer , TargetGroup
HTTPCode_Target_3XX_Count	HTTPCode_Target_3XX_Count	Sum	AvailabilityZone , LoadBalancer , TargetGroup
HTTPCode_Target_4XX_Count	HTTPCode_Target_4XX_Count	Sum	AvailabilityZone , LoadBalancer , TargetGroup
HTTPCode_Target_5XX_Count	HTTPCode_Target_5XX_Count	Sum	AvailabilityZone , LoadBalancer , TargetGroup
HTTP_Fixed_Response_Count	HTTP_Fixed_Response_Count	Sum	AvailabilityZone , LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HTTP_Redirect_Count	HTTP_Redirect_Count	Sum	AvailabilityZone , LoadBalancer
HTTP_Redirect_Url_Limit_Exceeded_Count	HTTP_Redirect_Url_Limit_Exceeded_Count	Sum	AvailabilityZone , LoadBalancer
HealthyHostCount	HealthyHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
HealthyStateDNS	HealthyStateDNS	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
HealthyStateRouting	HealthyStateRouting	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
IPv6ProcessedBytes	IPv6ProcessedBytes	Sum	AvailabilityZone , LoadBalancer
IPv6RequestCount	IPv6RequestCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LambdaTargetProcessedBytes	LambdaTargetProcessedBytes	Sum	LoadBalancer
LowReputationPacketsDropped	LowReputationPacketsDropped	Sum	AvailabilityZone , LoadBalancer
LowReputationRequestsDenied	LowReputationRequestsDenied	Sum	AvailabilityZone , LoadBalancer
MitigatedHostCount	MitigatedHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
NewConnectionCount	NewConnectionCount	Sum	AvailabilityZone , LoadBalancer
NonStickyRequestCount	NonStickyRequestCount	Sum	AvailabilityZone , LoadBalancer
PeakLCUs	PeakLCUs	Histogram	LoadBalancer
ProcessedBytes	ProcessedBytes	Sum	AvailabilityZone , LoadBalancer
RejectedConnectionCount	RejectedConnectionCount	Sum	AvailabilityZone , LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
RequestCount	RequestCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup
RequestCountPerTarget	RequestCountPerTarget	Sum	AvailabilityZone , LoadBalancer , TargetGroup
ReservedLCUs	ReservedLCUs	Histogram	LoadBalancer
RuleEvaluations	RuleEvaluations	Sum	LoadBalancer
TargetConnectionErrorCount	TargetConnectionErrorCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup
TargetControlActiveChannelCount	TargetControlActiveChannelCount	Histogram	AvailabilityZone , LoadBalancer
TargetControlChannelErrorCount	TargetControlChannelErrorCount	Sum	AvailabilityZone , LoadBalancer
TargetControlNewChannelCount	TargetControlNewChannelCount	Sum	AvailabilityZone , LoadBalancer
TargetControlProcessedBytes	TargetControlProcessedBytes	Sum	AvailabilityZone , LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TargetControlRequestCount	TargetControlRequestCount	Sum	AvailabilityZone , LoadBalancer
TargetControlRequestRejectCount	TargetControlRequestRejectCount	Sum	AvailabilityZone , LoadBalancer
TargetControlWorkQueueLength	TargetControlWorkQueueLength	Histogram	AvailabilityZone , LoadBalancer
TargetResponseTime	TargetResponseTime	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
TargetTLSNegotiationErrorCount	TargetTLSNegotiationErrorCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup
UnHealthyHostCount	UnHealthyHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
UnhealthyRoutingRequestCount	UnhealthyRoutingRequestCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
UnhealthyStateDNS	UnhealthyStateDNS	Histogram	AvailabilityZone, LoadBalancer, TargetGroup
UnhealthyStateRouting	UnhealthyStateRouting	Histogram	AvailabilityZone, LoadBalancer, TargetGroup

AWS::ElasticLoadBalancingV2::TargetGroup

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LambdaInternalError	LambdaInternalError	Sum	TargetGroup
LambdaUserError	LambdaUserError	Sum	TargetGroup

AWS/Athena

The AWS/Athena namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/athena`. All enriched instruments use Delta temporality.

- AWS::Athena::CapacityReservation

AWS::Athena::CapacityReservation

- `cloud.resource_id` enrichment: Supported

- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DPUConsumed	DPUConsumed	Histogram	Capacity Reservation , WorkGroup

AWS/AutoScaling

The `AWS/AutoScaling` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/autoscaling`. All enriched instruments use Delta temporality.

- AWS::AutoScaling::AutoScalingGroup

AWS::AutoScaling::AutoScalingGroup

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GroupAndWarmPoolDesiredCapacity	GroupAndWarmPoolDesiredCapacity	Histogram	AutoScalingGroupName
GroupAndWarmPoolTotalCapacity	GroupAndWarmPoolTotalCapacity	Histogram	AutoScalingGroupName
GroupDesiredCapacity	GroupDesiredCapacity	Histogram	AutoScalingGroupName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GroupInServiceCapacity	GroupInServiceCapacity	Histogram	AutoScalingGroupName
GroupInServiceInstances	GroupInServiceInstances	Histogram	AutoScalingGroupName
GroupMaxSize	GroupMaxSize	Histogram	AutoScalingGroupName
GroupMinSize	GroupMinSize	Histogram	AutoScalingGroupName
GroupPendingCapacity	GroupPendingCapacity	Histogram	AutoScalingGroupName
GroupPendingInstances	GroupPendingInstances	Histogram	AutoScalingGroupName
GroupStandbyCapacity	GroupStandbyCapacity	Histogram	AutoScalingGroupName
GroupStandbyInstances	GroupStandbyInstances	Histogram	AutoScalingGroupName
GroupTerminatingCapacity	GroupTerminatingCapacity	Histogram	AutoScalingGroupName
GroupTerminatingInstances	GroupTerminatingInstances	Histogram	AutoScalingGroupName
GroupTerminatingRetainedCapacity	GroupTerminatingRetainedCapacity	Histogram	AutoScalingGroupName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GroupTerminatingRetainedInstances	GroupTerminatingRetainedInstances	Histogram	AutoScalingGroupName
GroupTotalCapacity	GroupTotalCapacity	Histogram	AutoScalingGroupName
GroupTotalInstances	GroupTotalInstances	Histogram	AutoScalingGroupName
PredictiveScalingCapacityForecast	PredictiveScalingCapacityForecast	Histogram	AutoScalingGroupName , PolicyName
PredictiveScalingLoadForecast	PredictiveScalingLoadForecast	Histogram	AutoScalingGroupName , PairIndex , PolicyName
PredictiveScalingMetricPairCorrelation	PredictiveScalingMetricPairCorrelation	Histogram	AutoScalingGroupName , PairIndex , PolicyName
WarmPoolDesiredCapacity	WarmPoolDesiredCapacity	Histogram	AutoScalingGroupName
WarmPoolMinSize	WarmPoolMinSize	Histogram	AutoScalingGroupName
WarmPoolPendingCapacity	WarmPoolPendingCapacity	Histogram	AutoScalingGroupName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WarmPoolPendingRetainedCapacity	WarmPoolPendingRetainedCapacity	Histogram	AutoScalingGroupName
WarmPoolTerminatingCapacity	WarmPoolTerminatingCapacity	Histogram	AutoScalingGroupName
WarmPoolTerminatingRetainedCapacity	WarmPoolTerminatingRetainedCapacity	Histogram	AutoScalingGroupName
WarmPoolTotalCapacity	WarmPoolTotalCapacity	Histogram	AutoScalingGroupName
WarmPoolWarmedCapacity	WarmPoolWarmedCapacity	Histogram	AutoScalingGroupName

AWS/Backup

The AWS/Backup namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/backup. All enriched instruments use Delta temporality.

- AWS::Backup::BackupVault

AWS::Backup::BackupVault

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NumberOfBackupJobsCompleted	NumberOfBackupJobsCompleted	Sum	BackupVaultName , ResourceType
NumberOfBackupJobsCreated	NumberOfBackupJobsCreated	Sum	BackupVaultName , ResourceType
NumberOfBackupJobsExpired	NumberOfBackupJobsExpired	Sum	BackupVaultName , ResourceType
NumberOfBackupJobsFailed	NumberOfBackupJobsFailed	Sum	BackupVaultName , ResourceType
NumberOfBackupJobsPartial	NumberOfBackupJobsPartial	Sum	BackupVaultName , ResourceType
NumberOfBackupJobsPending	NumberOfBackupJobsPending	Histogram	BackupVaultName , ResourceType
NumberOfBackupJobsRunning	NumberOfBackupJobsRunning	Histogram	BackupVaultName , ResourceType
NumberOfCopyJobsCompleted	NumberOfCopyJobsCompleted	Sum	BackupVaultName , ResourceType
NumberOfCopyJobsCreated	NumberOfCopyJobsCreated	Sum	BackupVaultName , ResourceType

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NumberOfCopyJobsFailed	NumberOfCopyJobsFailed	Sum	BackupVaultName , ResourceType
NumberOfCopyJobsRunning	NumberOfCopyJobsRunning	Histogram	BackupVaultName , ResourceType
NumberOfRecoveryPointsCold	NumberOfRecoveryPointsCold	Histogram	BackupVaultName , ResourceType
NumberOfRecoveryPointsCompleted	NumberOfRecoveryPointsCompleted	Sum	BackupVaultName , ResourceType
NumberOfRecoveryPointsCreating	NumberOfRecoveryPointsCreating	Histogram	BackupVaultName , ResourceType
NumberOfRecoveryPointsDeleting	NumberOfRecoveryPointsDeleting	Histogram	BackupVaultName , ResourceType
NumberOfRecoveryPointsExpired	NumberOfRecoveryPointsExpired	Histogram	BackupVaultName , ResourceType
NumberOfRecoveryPointsPartial	NumberOfRecoveryPointsPartial	Histogram	BackupVaultName , ResourceType
NumberOfRestoreJobsCompleted	NumberOfRestoreJobsCompleted	Sum	BackupVaultName , ResourceType

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NumberOfRestoreJobsFailed	NumberOfRestoreJobsFailed	Sum	BackupVaultName , ResourceType
NumberOfRestoreJobsPending	NumberOfRestoreJobsPending	Histogram	BackupVaultName , ResourceType
NumberOfRestoreJobsRunning	NumberOfRestoreJobsRunning	Histogram	BackupVaultName , ResourceType

AWS/Bedrock

The AWS/Bedrock namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/bedrock. All enriched instruments use Delta temporality.

- model

model

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CacheReadInputTokens	CacheReadInputTokens	Sum	ModelId
CacheWriteInputTokens	CacheWriteInputTokens	Sum	ModelId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EstimatedTPMQuotaUsage	EstimatedTPMQuotaUsage	Histogram	ModelId
InputTokenCount	InputTokenCount	Sum	ModelId
InvocationClientErrors	InvocationClientErrors	Sum	ModelId
InvocationLatency	InvocationLatency	Histogram	ModelId
InvocationServerErrors	InvocationServerErrors	Sum	ModelId
InvocationThrottles	InvocationThrottles	Sum	ModelId
Invocations	Invocations	Sum	ModelId
LegacyModelInvocations	LegacyModelInvocations	Sum	ModelId
OutputImageCount	OutputImageCount	Sum	BucketedStepSize , ImageSize , ModelId
OutputTokenCount	OutputTokenCount	Sum	ModelId
TimeToFirstToken	TimeToFirstToken	Histogram	ModelId

AWS/BedrockMantle

The `AWS/BedrockMantle` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/bedrock-mantle`. All enriched instruments use Delta temporality.

- AWS::BedrockMantle::Project

AWS::BedrockMantle::Project

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`Inference ClientErrors`	`Inference ClientErrors`	Sum	`Model`, `Project`
`Inferences`	`Inferences`	Sum	`Model`, `Project`
`InputTokens`	`InputTokens`	Histogram	`Model`, `Project`
`OutputTokens`	`OutputTokens`	Histogram	`Model`, `Project`
`TotalInpu tTokens`	`TotalInpu tTokens`	Sum	`Project`
`TotalOutp utTokens`	`TotalOutp utTokens`	Sum	`Project`

AWS/ClientVPN

The `AWS/ClientVPN` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/ec2`. All enriched instruments use Delta temporality.

- AWS::EC2::ClientVpnEndpoint

AWS::EC2::ClientVpnEndpoint

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`ActiveConnectionsCount`	`ActiveConnectionsCount`	Histogram	`Endpoint`
`AuthenticationFailures`	`AuthenticationFailures`	Sum	`Endpoint`
`ClientConnectHandlerDeniedConnections`	`ClientConnectHandlerDeniedConnections`	Sum	`Endpoint`
`ClientConnectHandlerFailedServiceErrors`	`ClientConnectHandlerFailedServiceErrors`	Sum	`Endpoint`
`ClientConnectHandlerInvalidResponses`	`ClientConnectHandlerInvalidResponses`	Sum	`Endpoint`
`ClientConnectHandlerOtherExecutionErrors`	`ClientConnectHandlerOtherExecutionErrors`	Sum	`Endpoint`
`ClientConnectHandl`	`ClientConnectHandl`	Sum	`Endpoint`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
erThrottl ingErrors	erThrottl ingErrors		
ClientCon nectHandl erTimeouts	ClientCon nectHandl erTimeouts	Sum	Endpoint
CrlDaysToExpiry	CrlDaysToExpiry	Histogram	Endpoint
EgressBytes	EgressBytes	Sum	Endpoint
EgressPackets	EgressPackets	Sum	Endpoint
IngressBytes	IngressBytes	Sum	Endpoint
IngressPackets	IngressPackets	Sum	Endpoint
SelfServi cePortalC lientConf iguration Downloads	SelfServi cePortalC lientConf iguration Downloads	Sum	Endpoint
ZeroHealt hItemsCount	ZeroHealt hItemsCount	Histogram	Endpoint

AWS/CloudFront

The `AWS/CloudFront` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/cloudfront`. All enriched instruments use Delta temporality.

- AWS::CloudFront::Distribution
- AWS::CloudFront::Function

AWS::CloudFront::Distribution

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`401ErrorRate`	`401ErrorRate`	Histogram	`DistributionId` , `Region`
`403ErrorRate`	`403ErrorRate`	Histogram	`DistributionId` , `Region`
`404ErrorRate`	`404ErrorRate`	Histogram	`DistributionId` , `Region`
`4xxErrorRate`	`4xxErrorRate`	Histogram	`DistributionId` , `Region`
`5xxErrorRate`	`5xxErrorRate`	Histogram	`DistributionId` , `Region`
`BytesDownloaded`	`BytesDownloaded`	Sum	`DistributionId` , `Region`
`BytesUploaded`	`BytesUploaded`	Sum	`DistributionId` , `Region`
`CacheHitRate`	`CacheHitRate`	Histogram	`DistributionId` , `Region`
`OriginLatency`	`OriginLatency`	Histogram	`DistributionId` , `Region`
`Requests`	`Requests`	Sum	`DistributionId` , `Region`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalErrorRate	TotalErrorRate	Histogram	DistributionId , Region

AWS::CloudFront::Function

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FunctionComputeUtilization	FunctionComputeUtilization	Histogram	DistributionId , FunctionName , Region
FunctionExecutionErrors	FunctionExecutionErrors	Sum	DistributionId , FunctionName , Region
FunctionInvocations	FunctionInvocations	Sum	DistributionId , FunctionName , Region
FunctionThrottles	FunctionThrottles	Sum	DistributionId , FunctionName , Region
FunctionValidationErrors	FunctionValidationErrors	Sum	DistributionId , FunctionName , Region

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KvsReadErrors	KvsReadErrors	Sum	DistributionId , FunctionName , Region
KvsReadRequests	KvsReadRequests	Sum	DistributionId , FunctionName , Region

AWS/CloudWatch/MetricStreams

The AWS/CloudWatch/MetricStreams namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/cloudwatch. All enriched instruments use Delta temporality.

- AWS::CloudWatch::MetricStream

AWS::CloudWatch::MetricStream

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MetricUpdate	MetricUpdate	Sum	MetricStreamName
PublishErrorRate	PublishErrorRate	Histogram	MetricStreamName
TotalMetricUpdate	TotalMetricUpdate	Sum	MetricStreamName

AWS/CodeGuruProfiler

The `AWS/CodeGuruProfiler` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/codeguru-profiler`. All enriched instruments use Delta temporality.

- AWS::CodeGuruProfiler::ProfilingGroup

AWS::CodeGuruProfiler::ProfilingGroup

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`Recommendations`	`Recommendations`	Histogram	`ProfilingGroupName`

AWS/Cognito

The `AWS/Cognito` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/cognito-idp`. All enriched instruments use Delta temporality.

- AWS::Cognito::UserPool

AWS::Cognito::UserPool

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AccountTakeoverRisk	AccountTakeoverRisk	Sum	Operation , RiskLevel , UserPoolId
CompromisedCredentialRisk	CompromisedCredentialRisk	Sum	Operation , RiskLevel , UserPoolId
FederationSuccesses	FederationSuccesses	Sum	IdentityProvider , UserPool, UserPoolClient
FederationThrottles	FederationThrottles	Sum	IdentityProvider , UserPool, UserPoolClient
NoRisk	NoRisk	Sum	Operation , UserPoolId
Risk	Risk	Sum	Operation , UserPoolId
SignInSuccesses	SignInSuccesses	Sum	UserPool, UserPoolClient
SignInThrottles	SignInThrottles	Sum	UserPool, UserPoolClient
SignUpSuccesses	SignUpSuccesses	Sum	UserPool, UserPoolClient
SignUpThrottles	SignUpThrottles	Sum	UserPool, UserPoolClient

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TokenRefreshSuccesses	TokenRefreshSuccesses	Sum	UserPool, UserPoolClient
TokenRefreshThrottles	TokenRefreshThrottles	Sum	UserPool, UserPoolClient

AWS/Connect

The AWS/Connect namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/connect. All enriched instruments use Delta temporality.

- AWS::Connect::Instance
- contact

AWS::Connect::Instance

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CallRecordingUploadError	CallRecordingUploadError	Sum	InstanceId , MetricGroup
CallsBreachingConcurrencyQuota	CallsBreachingConcurrencyQuota	Sum	InstanceId , MetricGroup
CallsPerInterval	CallsPerInterval	Sum	InstanceId , MetricGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ChatsBreachingActiveChatQuota	ChatsBreachingActiveChatQuota	Sum	InstanceId , MetricGroup
ConcurrentActiveChats	ConcurrentActiveChats	Histogram	InstanceId , MetricGroup
ConcurrentActiveChatsPercentage	ConcurrentActiveChatsPercentage	Histogram	InstanceId , MetricGroup
ConcurrentCalls	ConcurrentCalls	Histogram	InstanceId , MetricGroup
ConcurrentCallsPercentage	ConcurrentCallsPercentage	Histogram	InstanceId , MetricGroup
ConcurrentTasks	ConcurrentTasks	Histogram	InstanceId , MetricGroup
ConcurrentTasksPercentage	ConcurrentTasksPercentage	Histogram	InstanceId , MetricGroup
ContactFlowErrors	ContactFlowErrors	Sum	ContactFlowName , InstanceId , MetricGroup
LongestQueueWaitTime	LongestQueueWaitTime	Histogram	InstanceId , MetricGroup , QueueName
MisconfiguredPhoneNumbers	MisconfiguredPhoneNumbers	Sum	InstanceId , MetricGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MissedCalls	MissedCalls	Sum	InstanceId , MetricGroup
PublicSigningKeyUsage	PublicSigningKeyUsage	Sum	InstanceId , SigningKeyId
QueueCapacityExceededError	QueueCapacityExceededError	Sum	InstanceId , MetricGroup , QueueName
QueueSize	QueueSize	Histogram	InstanceId , MetricGroup , QueueName
SuccessfulChatsPerInterval	SuccessfulChatsPerInterval	Sum	InstanceId , MetricGroup
TasksBreachingConcurrencyQuota	TasksBreachingConcurrencyQuota	Sum	InstanceId , MetricGroup
ToInstancePacketLossRate	ToInstancePacketLossRate	Histogram	InstanceId , Participant , Stream Type, Type of Connection

contact

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TasksExpired	TasksExpired	Sum	ContactId , InstanceId , MetricGroup
TasksExpiryWarningReached	TasksExpiryWarningReached	Sum	ContactId , InstanceId , MetricGroup

AWS/DAX

The AWS/DAX namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/dax. All enriched instruments use Delta temporality.

- AWS::DAX::Cluster

AWS::DAX::Cluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BaselineNetworkBytesInUtilization	BaselineNetworkBytesInUtilization	Histogram	ClusterId , NodeId
BaselineNetworkBytesOutUtilization	BaselineNetworkBytesOutUtilization	Histogram	ClusterId , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BatchGetItemRequestCount	BatchGetItemRequestCount	Sum	ClusterId , NodeId
BatchWriteItemRequestCount	BatchWriteItemRequestCount	Sum	ClusterId , NodeId
CPUCreditBalance	CPUCreditBalance	Histogram	ClusterId , NodeId
CPUCreditUsage	CPUCreditUsage	Histogram	ClusterId , NodeId
CPUSurplusCreditBalance	CPUSurplusCreditBalance	Histogram	ClusterId , NodeId
CPUSurplusCreditsCharged	CPUSurplusCreditsCharged	Sum	ClusterId , NodeId
CPUUtilization	CPUUtilization	Histogram	ClusterId , NodeId
CacheMemoryUtilization	CacheMemoryUtilization	Histogram	ClusterId , NodeId
ClientConnections	ClientConnections	Histogram	ClusterId , NodeId
DeleteItemRequestCount	DeleteItemRequestCount	Sum	ClusterId , NodeId
ErrorRequestCount	ErrorRequestCount	Sum	ClusterId , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EstimatedDbSize	EstimatedDbSize	Histogram	ClusterId , NodeId
EvictedSize	EvictedSize	Histogram	ClusterId , NodeId
FailedRequestCount	FailedRequestCount	Sum	ClusterId , NodeId
FaultRequestCount	FaultRequestCount	Sum	ClusterId , NodeId
GetItemRequestCount	GetItemRequestCount	Sum	ClusterId , NodeId
ItemCacheHits	ItemCacheHits	Sum	ClusterId , NodeId
ItemCacheMisses	ItemCacheMisses	Sum	ClusterId , NodeId
NetworkBytesIn	NetworkBytesIn	Sum	ClusterId , NodeId
NetworkBytesOut	NetworkBytesOut	Sum	ClusterId , NodeId
NetworkMaxBytesIn	NetworkMaxBytesIn	Histogram	ClusterId , NodeId
NetworkMaxBytesOut	NetworkMaxBytesOut	Histogram	ClusterId , NodeId
NetworkMaxPacketsIn	NetworkMaxPacketsIn	Histogram	ClusterId , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkMaxPacketsOut	NetworkMaxPacketsOut	Histogram	ClusterId , NodeId
NetworkPacketsIn	NetworkPacketsIn	Sum	ClusterId , NodeId
NetworkPacketsOut	NetworkPacketsOut	Sum	ClusterId , NodeId
PutItemRequestCount	PutItemRequestCount	Sum	ClusterId , NodeId
QueryCacheHits	QueryCacheHits	Sum	ClusterId , NodeId
QueryCacheMisses	QueryCacheMisses	Sum	ClusterId , NodeId
QueryRequestCount	QueryRequestCount	Sum	ClusterId , NodeId
ScanCacheHits	ScanCacheHits	Sum	ClusterId , NodeId
ScanCacheMisses	ScanCacheMisses	Sum	ClusterId , NodeId
ScanRequestCount	ScanRequestCount	Sum	ClusterId , NodeId
ThrottledRequestCount	ThrottledRequestCount	Sum	ClusterId , NodeId
TotalRequestCount	TotalRequestCount	Sum	ClusterId , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TransactGetItemsCount	TransactGetItemsCount	Sum	ClusterId , NodeId
TransactWriteItemsCount	TransactWriteItemsCount	Sum	ClusterId , NodeId
UpdateItemRequestCount	UpdateItemRequestCount	Sum	ClusterId , NodeId

AWS/DX

The AWS/DX namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/directconnect`. All enriched instruments use Delta temporality.

- AWS::DirectConnect::VirtualInterface
- dxcon

AWS::DirectConnect::VirtualInterface

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
VirtualInterfaceBgpPrefixesAccepted	VirtualInterfaceBgpPrefixesAccepted	Histogram	ConnectionId , IpAddressFamily , VirtualInterfaceId
VirtualInterfaceBg	VirtualInterfaceBg	Histogram	ConnectionId , IpAddressFamily ,

Metric	OpenTelemetry metric	Instrument type	Data point attributes
pPrefixes Advertised	pPrefixes Advertised		VirtualIn terfaceId
VirtualIn terfaceBg pStatus	VirtualIn terfaceBg pStatus	Histogram	ConnectionId , IpAddressFamily , VirtualIn terfaceId
VirtualIn terfaceBp sEgress	VirtualIn terfaceBp sEgress	Histogram	Connectio nId , VirtualIn terfaceId
VirtualIn terfaceBp sIngress	VirtualIn terfaceBp sIngress	Histogram	Connectio nId , VirtualIn terfaceId
VirtualIn terfacePp sEgress	VirtualIn terfacePp sEgress	Histogram	Connectio nId , VirtualIn terfaceId
VirtualIn terfacePp sIngress	VirtualIn terfacePp sIngress	Histogram	Connectio nId , VirtualIn terfaceId

dxcon

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConnectionBpsEgress	ConnectionBpsEgress	Histogram	ConnectionId
ConnectionBpsIngress	ConnectionBpsIngress	Histogram	ConnectionId
ConnectionCRCErrorCount	ConnectionCRCErrorCount	Sum	ConnectionId
ConnectionEncryptionState	ConnectionEncryptionState	Histogram	ConnectionId
ConnectionErrorCount	ConnectionErrorCount	Sum	ConnectionId
ConnectionLightLevelRx	ConnectionLightLevelRx	Histogram	ConnectionId ,OpticalLaneNumber
ConnectionLightLevelTx	ConnectionLightLevelTx	Histogram	ConnectionId ,OpticalLaneNumber
ConnectionPpsEgress	ConnectionPpsEgress	Histogram	ConnectionId
ConnectionPpsIngress	ConnectionPpsIngress	Histogram	ConnectionId
ConnectionState	ConnectionState	Histogram	ConnectionId

AWS/DataSync

The `AWS/DataSync` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/datasync`. All enriched instruments use Delta temporality.

- AWS::DataSync::Agent
- AWS::DataSync::Task

AWS::DataSync::Agent

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`BytesCompressed`	`BytesCompressed`	Sum	`AgentId`
`BytesTransferred`	`BytesTransferred`	Sum	`AgentId`
`BytesWritten`	`BytesWritten`	Sum	`AgentId`
`FilesTransferred`	`FilesTransferred`	Sum	`AgentId`

AWS::DataSync::Task

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`BytesCompressed`	`BytesCompressed`	Sum	`TaskId`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesPreparedDestination	BytesPreparedDestination	Sum	TaskId
BytesPreparedSource	BytesPreparedSource	Sum	TaskId
BytesTransferred	BytesTransferred	Sum	TaskId
BytesVerifiedDestination	BytesVerifiedDestination	Sum	TaskId
BytesVerifiedSource	BytesVerifiedSource	Sum	TaskId
BytesWritten	BytesWritten	Sum	TaskId
FilesDeleted	FilesDeleted	Sum	TaskId
FilesListedSource	FilesListedSource	Sum	TaskId
FilesPrepared	FilesPrepared	Sum	TaskId
FilesPreparedDestination	FilesPreparedDestination	Sum	TaskId
FilesPreparedSource	FilesPreparedSource	Sum	TaskId
FilesSkipped	FilesSkipped	Sum	TaskId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FilesTransferred	FilesTransferred	Sum	TaskId
FilesVerified	FilesVerified	Sum	TaskId
FilesVerifiedDestination	FilesVerifiedDestination	Sum	TaskId
FilesVerifiedSource	FilesVerifiedSource	Sum	TaskId

AWS/DocDB

The AWS/DocDB namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/rds. All enriched instruments use Delta temporality.

- cluster
- instance

cluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AvailableMVCCIds	AvailableMVCCIds	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AvailableMVCCIds	AvailableMVCCIdsByRole	Histogram	DBClusterIdentifier, Role
BackupRetentionPeriodStorageUsed	BackupRetentionPeriodStorageUsed	Histogram	DBClusterIdentifier
BufferCacheHitRatio	BufferCacheHitRatio	Histogram	DBClusterIdentifier
BufferCacheHitRatio	BufferCacheHitRatioByRole	Histogram	DBClusterIdentifier, Role
CPUCreditBalance	CPUCreditBalance	Histogram	DBClusterIdentifier
CPUCreditBalance	CPUCreditBalanceByRole	Histogram	DBClusterIdentifier, Role
CPUCreditUsage	CPUCreditUsage	Histogram	DBClusterIdentifier
CPUCreditUsage	CPUCreditUsageByRole	Histogram	DBClusterIdentifier, Role
CPUSurplusCreditBalance	CPUSurplusCreditBalance	Histogram	DBClusterIdentifier
CPUSurplusCreditBalance	CPUSurplusCreditBalanceByRole	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUSurplusCreditsCharged	CPUSurplusCreditsCharged	Sum	DBClusterIdentifier
CPUSurplusCreditsCharged	CPUSurplusCreditsChargedByRole	Sum	DBClusterIdentifier, Role
CPUUtilization	CPUUtilization	Histogram	DBClusterIdentifier
CPUUtilization	CPUUtilizationByRole	Histogram	DBClusterIdentifier, Role
ChangeStreamLogSize	ChangeStreamLogSize	Histogram	DBClusterIdentifier
DBClusterReplicaLagMaximum	DBClusterReplicaLagMaximum	Histogram	DBClusterIdentifier
DBClusterReplicaLagMinimum	DBClusterReplicaLagMinimum	Histogram	DBClusterIdentifier
DBInstanceReplicaLag	DBInstanceReplicaLag	Histogram	DBClusterIdentifier
DBInstanceReplicaLag	DBInstanceReplicaLagByRole	Histogram	DBClusterIdentifier, Role
DatabaseConnections	DatabaseConnections	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DatabaseConnections	DatabaseConnectionsByRole	Histogram	DBClusterIdentifier, Role
DatabaseConnectionsActiveMax	DatabaseConnectionsActiveMax	Histogram	DBClusterIdentifier
DatabaseConnectionsActiveMax	DatabaseConnectionsActiveMaxByRole	Histogram	DBClusterIdentifier, Role
DatabaseConnectionsLimit	DatabaseConnectionsLimit	Histogram	DBClusterIdentifier
DatabaseConnectionsLimit	DatabaseConnectionsLimitByRole	Histogram	DBClusterIdentifier, Role
DatabaseConnectionsMax	DatabaseConnectionsMax	Histogram	DBClusterIdentifier
DatabaseConnectionsMax	DatabaseConnectionsMaxByRole	Histogram	DBClusterIdentifier, Role
DatabaseCursors	DatabaseCursors	Histogram	DBClusterIdentifier
DatabaseCursors	DatabaseCursorsByRole	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DatabaseCursorsLimit	DatabaseCursorsLimit	Histogram	DBClusterIdentifier
DatabaseCursorsLimit	DatabaseCursorsLimitByRole	Histogram	DBClusterIdentifier, Role
DatabaseCursorsMax	DatabaseCursorsMax	Histogram	DBClusterIdentifier
DatabaseCursorsMax	DatabaseCursorsMaxByRole	Histogram	DBClusterIdentifier, Role
DatabaseCursorsTimedOut	DatabaseCursorsTimedOut	Sum	DBClusterIdentifier
DatabaseCursorsTimedOut	DatabaseCursorsTimedOutByRole	Sum	DBClusterIdentifier, Role
DiagnosticAlertCount	DiagnosticAlertCount	Histogram	DBClusterIdentifier
DiagnosticAlertCount	DiagnosticAlertCountByRole	Histogram	DBClusterIdentifier, Role
DiskQueueDepth	DiskQueueDepth	Histogram	DBClusterIdentifier
DiskQueueDepth	DiskQueueDepthByRole	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DocumentsDeleted	documents.deleted	Sum	DBClusterIdentifier
DocumentsDeleted	documents.deleted.by_role	Sum	DBClusterIdentifier, Role
DocumentsInserted	documents.inserted	Sum	DBClusterIdentifier
DocumentsInserted	documents.inserted.by_role	Sum	DBClusterIdentifier, Role
DocumentsReturned	documents.returned	Sum	DBClusterIdentifier
DocumentsReturned	documents.returned.by_role	Sum	DBClusterIdentifier, Role
DocumentsUpdated	documents.updated	Sum	DBClusterIdentifier
DocumentsUpdated	documents.updated.by_role	Sum	DBClusterIdentifier, Role
EBSByteBalance%	ebs.byte_balance.percent	Histogram	DBClusterIdentifier
EBSByteBalance%	ebs.byte_balance.percent.by_role	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EBSIOBalance%	EBSIOBalance%	Histogram	DBCluster Identifier
EBSIOBalance%	EBSIOBalancePercentByRole	Histogram	DBCluster Identifier , Role
EngineUptime	EngineUptime	Histogram	DBCluster Identifier
EngineUptime	EngineUptimeByRole	Histogram	DBCluster Identifier , Role
FreeLocal Storage	FreeLocal Storage	Histogram	DBCluster Identifier
FreeLocal Storage	FreeLocal StorageByRole	Histogram	DBCluster Identifier , Role
FreeNVMeStorage	FreeNVMeStorageByRole	Histogram	DBCluster Identifier , Role
FreeableMemory	FreeableMemory	Histogram	DBCluster Identifier
FreeableMemory	FreeableMemoryByRole	Histogram	DBCluster Identifier , Role
IndexBufferCacheHitRatio	IndexBufferCacheHitRatio	Histogram	DBCluster Identifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
IndexBufferCacheHitRatio	IndexBufferCacheHitRatioByRole	Histogram	DBClusterIdentifier, Role
LongestActiveGCRuntime	LongestActiveGCRuntime	Histogram	DBClusterIdentifier
LongestActiveGCRuntime	LongestActiveGCRuntimeByRole	Histogram	DBClusterIdentifier, Role
LowMemNumOperationsThrottled	LowMemNumOperationsThrottled	Sum	DBClusterIdentifier
LowMemNumOperationsThrottled	LowMemNumOperationsThrottledByRole	Sum	DBClusterIdentifier, Role
LowMemNumOperationsTimedOut	LowMemNumOperationsTimedOut	Sum	DBClusterIdentifier
LowMemNumOperationsTimedOut	LowMemNumOperationsTimedOutByRole	Sum	DBClusterIdentifier, Role
LowMemThrottleMaxQueueDepth	LowMemThrottleMaxQueueDepth	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LowMemThrottleMaxQueueDepth	LowMemThrottleMaxQueueDepthByRole	Histogram	DBClusterIdentifier, Role
LowMemThrottleQueueDepth	LowMemThrottleQueueDepth	Histogram	DBClusterIdentifier
LowMemThrottleQueueDepth	LowMemThrottleQueueDepthByRole	Histogram	DBClusterIdentifier, Role
NVMeStorageCacheHitRatio	NVMeStorageCacheHitRatioByRole	Histogram	DBClusterIdentifier, Role
NetworkReceiveThroughput	NetworkReceiveThroughput	Histogram	DBClusterIdentifier
NetworkReceiveThroughput	NetworkReceiveThroughputByRole	Histogram	DBClusterIdentifier, Role
NetworkThroughput	NetworkThroughput	Histogram	DBClusterIdentifier
NetworkThroughput	NetworkThroughputByRole	Histogram	DBClusterIdentifier, Role
NetworkTransmitThroughput	NetworkTransmitThroughput	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkTransmitThroughput	NetworkTransmitThroughputByRole	Histogram	DBClusterIdentifier, Role
OpcountersCommand	OpcountersCommand	Sum	DBClusterIdentifier
OpcountersCommand	OpcountersCommandByRole	Sum	DBClusterIdentifier, Role
OpcountersDelete	OpcountersDelete	Sum	DBClusterIdentifier
OpcountersDelete	OpcountersDeleteByRole	Sum	DBClusterIdentifier, Role
OpcountersGetmore	OpcountersGetmore	Sum	DBClusterIdentifier
OpcountersGetmore	OpcountersGetmoreByRole	Sum	DBClusterIdentifier, Role
OpcountersInsert	OpcountersInsert	Sum	DBClusterIdentifier
OpcountersInsert	OpcountersInsertByRole	Sum	DBClusterIdentifier, Role
OpcountersQuery	OpcountersQuery	Sum	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OpcountersQuery	Opcounter sQueryByRole	Sum	DBCluster Identifier , Role
Opcounter sUpdate	Opcounter sUpdate	Sum	DBCluster Identifier
Opcounter sUpdate	Opcounter sUpdateByRole	Sum	DBCluster Identifier , Role
ReadIOPS	ReadIOPS	Histogram	DBCluster Identifier
ReadIOPS	ReadIOPSByRole	Histogram	DBCluster Identifier , Role
ReadIOPSN VMeStorage	ReadIOPSN VMeStorag eByRole	Histogram	DBCluster Identifier , Role
ReadLatency	ReadLatency	Histogram	DBCluster Identifier
ReadLatency	ReadLaten cyByRole	Histogram	DBCluster Identifier , Role
ReadLaten cyNVMeStorage	ReadLaten cyNVMeSto rageByRole	Histogram	DBCluster Identifier , Role
ReadThroughput	ReadThroughput	Histogram	DBCluster Identifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReadThroughput	ReadThroughputByRole	Histogram	DBClusterIdentifier, Role
ReadThroughputNVMeStorage	ReadThroughputNVMeStorageByRole	Histogram	DBClusterIdentifier, Role
SnapshotStorageUsed	SnapshotStorageUsed	Histogram	DBClusterIdentifier
StorageNetworkReceiveThroughput	StorageNetworkReceiveThroughput	Histogram	DBClusterIdentifier
StorageNetworkReceiveThroughput	StorageNetworkReceiveThroughputByRole	Histogram	DBClusterIdentifier, Role
StorageNetworkThroughput	StorageNetworkThroughput	Histogram	DBClusterIdentifier
StorageNetworkThroughput	StorageNetworkThroughputByRole	Histogram	DBClusterIdentifier, Role
StorageNetworkTransmitThroughput	StorageNetworkTransmitThroughput	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StorageNetworkTransmitThroughput	StorageNetworkTransmitThroughputByRole	Histogram	DBClusterIdentifier, Role
StsGetCallerIdentityCalls	StsGetCallerIdentityCalls	Histogram	DBClusterIdentifier
StsGetCallerIdentityCalls	StsGetCallerIdentityCallsByRole	Histogram	DBClusterIdentifier, Role
StsMaxConcurrentRequests	StsMaxConcurrentRequests	Histogram	DBClusterIdentifier
StsMaxConcurrentRequests	StsMaxConcurrentRequestsByRole	Histogram	DBClusterIdentifier, Role
SwapUsage	SwapUsage	Histogram	DBClusterIdentifier
SwapUsage	SwapUsageByRole	Histogram	DBClusterIdentifier, Role
TTLDeletedDocuments	TTLDeletedDocuments	Sum	DBClusterIdentifier
TTLDeletedDocuments	TTLDeletedDocumentsByRole	Sum	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalBackupStorageBilled	TotalBackupStorageBilled	Histogram	DBClusterIdentifier
TransactionsAborted	TransactionsAborted	Sum	DBClusterIdentifier
TransactionsAborted	TransactionsAbortedByRole	Sum	DBClusterIdentifier, Role
TransactionsCommitted	TransactionsCommitted	Sum	DBClusterIdentifier
TransactionsCommitted	TransactionsCommittedByRole	Sum	DBClusterIdentifier, Role
TransactionsOpen	TransactionsOpen	Histogram	DBClusterIdentifier
TransactionsOpen	TransactionsOpenByRole	Histogram	DBClusterIdentifier, Role
TransactionsOpenLimit	TransactionsOpenLimit	Histogram	DBClusterIdentifier
TransactionsOpenLimit	TransactionsOpenLimitByRole	Histogram	DBClusterIdentifier, Role
TransactionsOpenMax	TransactionsOpenMax	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TransactionsOpenMax	TransactionsOpenMaxByRole	Histogram	DBClusterIdentifier, Role
TransactionsStarted	TransactionsStarted	Sum	DBClusterIdentifier
TransactionsStarted	TransactionsStartedByRole	Sum	DBClusterIdentifier, Role
VolumeBytesUsed	VolumeBytesUsed	Histogram	DBClusterIdentifier
VolumeReadIOPs	VolumeReadIOPs	Histogram	DBClusterIdentifier
VolumeWriteIOPs	VolumeWriteIOPs	Histogram	DBClusterIdentifier
WriteIOPS	WriteIOPS	Histogram	DBClusterIdentifier
WriteIOPS	WriteIOPSByRole	Histogram	DBClusterIdentifier, Role
WriteIOPSNVMeStorage	WriteIOPSNVMeStorageByRole	Histogram	DBClusterIdentifier, Role
WriteLatency	WriteLatency	Histogram	DBClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WriteLatency	WriteLatencyByRole	Histogram	DBClusterIdentifier, Role
WriteLatencyNVMeStorage	WriteLatencyNVMeStorageByRole	Histogram	DBClusterIdentifier, Role
WriteThroughput	WriteThroughput	Histogram	DBClusterIdentifier
WriteThroughput	WriteThroughputByRole	Histogram	DBClusterIdentifier, Role
WriteThroughputNVMeStorage	WriteThroughputNVMeStorageByRole	Histogram	DBClusterIdentifier, Role

`instance`

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AvailableMVCCIds	AvailableMVCCIdsByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BufferCacheHitRatio	BufferCacheHitRatioByInstance	Histogram	DBInstanceIdentifier
CPUCreditBalance	CPUCreditBalanceByInstance	Histogram	DBInstanceIdentifier
CPUCreditUsage	CPUCreditUsageByInstance	Histogram	DBInstanceIdentifier
CPUSurplusCreditBalance	CPUSurplusCreditBalanceByInstance	Histogram	DBInstanceIdentifier
CPUSurplusCreditsCharged	CPUSurplusCreditsChargedByInstance	Sum	DBInstanceIdentifier
CPUUtilization	CPUUtilizationByInstance	Histogram	DBInstanceIdentifier
DBInstanceReplicaLag	DBInstanceReplicaLagByInstance	Histogram	DBInstanceIdentifier
DatabaseConnections	DatabaseConnectionsByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DatabaseConnectionsActiveMax	DatabaseConnectionsActiveMaxByInstance	Histogram	DBInstanceIdentifier
DatabaseConnectionsLimit	DatabaseConnectionsLimitByInstance	Histogram	DBInstanceIdentifier
DatabaseConnectionsMax	DatabaseConnectionsMaxByInstance	Histogram	DBInstanceIdentifier
DatabaseCursors	DatabaseCursorsByInstance	Histogram	DBInstanceIdentifier
DatabaseCursorsLimit	DatabaseCursorsLimitByInstance	Histogram	DBInstanceIdentifier
DatabaseCursorsMax	DatabaseCursorsMaxByInstance	Histogram	DBInstanceIdentifier
DatabaseCursorsTimedOut	DatabaseCursorsTimedOutByInstance	Sum	DBInstanceIdentifier
DiagnosticAlertCount	DiagnosticAlertCountByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DiskQueueDepth	DiskQueueDepthByInstance	Histogram	DBInstanceIdentifier
DocumentsDeleted	DocumentsDeletedByInstance	Sum	DBInstanceIdentifier
DocumentsInserted	DocumentsInsertedByInstance	Sum	DBInstanceIdentifier
DocumentsReturned	DocumentsReturnedByInstance	Sum	DBInstanceIdentifier
DocumentsUpdated	DocumentsUpdatedByInstance	Sum	DBInstanceIdentifier
EBSByteBalance%	EBSByteBalancePercentByInstance	Histogram	DBInstanceIdentifier
EBSIOBalance%	EBSIOBalancePercentByInstance	Histogram	DBInstanceIdentifier
EngineUptime	EngineUptimeByInstance	Histogram	DBInstanceIdentifier
FreeLocalStorage	FreeLocalStorageByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FreeNVMeStorage	FreeNVMeStorageByInstance	Histogram	DBInstanceIdentifier
FreeableMemory	FreeableMemoryByInstance	Histogram	DBInstanceIdentifier
IndexBufferCacheHitRatio	IndexBufferCacheHitRatioByInstance	Histogram	DBInstanceIdentifier
LongestActiveGCRuntime	LongestActiveGCRuntimeByInstance	Histogram	DBInstanceIdentifier
LowMemNumOperationsThrottled	LowMemNumOperationsThrottledByInstance	Sum	DBInstanceIdentifier
LowMemNumOperationsTimedOut	LowMemNumOperationsTimedOutByInstance	Sum	DBInstanceIdentifier
LowMemThrottleMaxQueueDepth	LowMemThrottleMaxQueueDepthByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LowMemThrottleQueueDepth	LowMemThrottleQueueDepthByInstance	Histogram	DBInstanceIdentifier
NVMeStorageCacheHitRatio	NVMeStorageCacheHitRatioByInstance	Histogram	DBInstanceIdentifier
NetworkReceiveThroughput	NetworkReceiveThroughputByInstance	Histogram	DBInstanceIdentifier
NetworkThroughput	NetworkThroughputByInstance	Histogram	DBInstanceIdentifier
NetworkTransmitThroughput	NetworkTransmitThroughputByInstance	Histogram	DBInstanceIdentifier
OpcountersCommand	OpcountersCommandByInstance	Sum	DBInstanceIdentifier
OpcountersDelete	OpcountersDeleteByInstance	Sum	DBInstanceIdentifier
OpcountersGetmore	OpcountersGetmoreByInstance	Sum	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Opcounter sInsert	Opcounter sInsertBy Instance	Sum	DBInstanc eIdentifier
OpcountersQuery	Opcounter sQueryByI nstance	Sum	DBInstanc eIdentifier
Opcounter sUpdate	Opcounter sUpdateBy Instance	Sum	DBInstanc eIdentifier
ReadIOPS	ReadIOPSB yInstance	Histogram	DBInstanc eIdentifier
ReadIOPSN VMeStorage	ReadIOPSN VMeStorag eByInstance	Histogram	DBInstanc eIdentifier
ReadLatency	ReadLaten cyByInstance	Histogram	DBInstanc eIdentifier
ReadLaten cyNVMeStorage	ReadLaten cyNVMeSto rageByInstance	Histogram	DBInstanc eIdentifier
ReadThroughput	ReadThrou ghputByIn stance	Histogram	DBInstanc eIdentifier
ReadThrou ghputNVMe Storage	ReadThrou ghputNVMe StorageBy Instance	Histogram	DBInstanc eIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StorageNetworkReceiveThroughput	StorageNetworkReceiveThroughputByInstance	Histogram	DBInstanceIdentifier
StorageNetworkThroughput	StorageNetworkThroughputByInstance	Histogram	DBInstanceIdentifier
StorageNetworkTransmitThroughput	StorageNetworkTransmitThroughputByInstance	Histogram	DBInstanceIdentifier
StsGetCallerIdentityCalls	StsGetCallerIdentityCallsByInstance	Histogram	DBInstanceIdentifier
StsMaxConcurrentRequests	StsMaxConcurrentRequestsByInstance	Histogram	DBInstanceIdentifier
SwapUsage	SwapUsageByInstance	Histogram	DBInstanceIdentifier
TTLDeletedDocuments	TTLDeletedDocumentsByInstance	Sum	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TransactionsAborted	TransactionsAbortedByInstance	Sum	DBInstanceIdentifier
TransactionsCommitted	TransactionsCommittedByInstance	Sum	DBInstanceIdentifier
TransactionsOpen	TransactionsOpenByInstance	Histogram	DBInstanceIdentifier
TransactionsOpenLimit	TransactionsOpenLimitByInstance	Histogram	DBInstanceIdentifier
TransactionsOpenMax	TransactionsOpenMaxByInstance	Histogram	DBInstanceIdentifier
TransactionsStarted	TransactionsStartedByInstance	Sum	DBInstanceIdentifier
WriteIOPS	WriteIOPSByInstance	Histogram	DBInstanceIdentifier
WriteIOPSNVMeStorage	WriteIOPSNVMeStorageByInstance	Histogram	DBInstanceIdentifier
WriteLatency	WriteLatencyByInstance	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WriteLatencyNVMeStorage	WriteLatencyNVMeStorageByInstance	Histogram	DBInstanceIdentifier
WriteThroughput	WriteThroughputByInstance	Histogram	DBInstanceIdentifier
WriteThroughputNVMeStorage	WriteThroughputNVMeStorageByInstance	Histogram	DBInstanceIdentifier

AWS/DocDB-Elastic

The AWS/DocDB-Elastic namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/docdb-elastic. All enriched instruments use Delta temporality.

- AWS::DocDBElastic::Cluster

AWS::DocDBElastic::Cluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BufferCacheHitRatio	BufferCacheHitRatio	Histogram	ClusterId , ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BufferCacheHitRatio	BufferCacheHitRatioByShard	Histogram	ClusterId , ClusterName , ShardId
DatabaseConnections	DatabaseConnections	Histogram	ClusterId , ClusterName
DatabaseConnections	DatabaseConnectionsByShard	Histogram	ClusterId , ClusterName , ShardId
DatabaseCursorsMax	DatabaseCursorsMax	Histogram	ClusterId , ClusterName
DatabaseCursorsMax	DatabaseCursorsMaxByShard	Histogram	ClusterId , ClusterName , ShardId
DatabaseCursorsTimedOut	DatabaseCursorsTimedOut	Sum	ClusterId , ClusterName
DatabaseCursorsTimedOut	DatabaseCursorsTimedOutByShard	Sum	ClusterId , ClusterName , ShardId
DocumentsDeleted	DocumentsDeleted	Sum	ClusterId , ClusterName
DocumentsDeleted	DocumentsDeletedByShard	Sum	ClusterId , ClusterName , ShardId
DocumentsInserted	DocumentsInserted	Sum	ClusterId , ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DocumentsInserted	DocumentsInsertedByShard	Sum	ClusterId , ClusterName , ShardId
DocumentsReturned	DocumentsReturned	Sum	ClusterId , ClusterName
DocumentsReturned	DocumentsReturnedByShard	Sum	ClusterId , ClusterName , ShardId
DocumentsUpdated	DocumentsUpdated	Sum	ClusterId , ClusterName
DocumentsUpdated	DocumentsUpdatedByShard	Sum	ClusterId , ClusterName , ShardId
OpcountersCommand	OpcountersCommand	Sum	ClusterId , ClusterName
OpcountersCommand	OpcountersCommandByShard	Sum	ClusterId , ClusterName , ShardId
OpcountersDelete	OpcountersDelete	Sum	ClusterId , ClusterName
OpcountersDelete	OpcountersDeleteByShard	Sum	ClusterId , ClusterName , ShardId
OpcountersGetmore	OpcountersGetmore	Sum	ClusterId , ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Opcounters Getmore	Opcounters GetmoreByShard	Sum	ClusterId , ClusterName , ShardId
Opcounters Insert	Opcounters Insert	Sum	ClusterId , ClusterName
Opcounters Insert	Opcounters InsertByShard	Sum	ClusterId , ClusterName , ShardId
OpcountersQuery	OpcountersQuery	Sum	ClusterId , ClusterName
OpcountersQuery	Opcounters QueryByShard	Sum	ClusterId , ClusterName , ShardId
Opcounters Update	Opcounters Update	Sum	ClusterId , ClusterName
Opcounters Update	Opcounters UpdateByShard	Sum	ClusterId , ClusterName , ShardId
PrimaryIn stanceCPU Utilization	PrimaryIn stanceCPU Utilization	Histogram	ClusterId , ClusterName
PrimaryIn stanceCPU Utilization	PrimaryIn stanceCPU Utilizati onByShard	Histogram	ClusterId , ClusterName , ShardId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PrimaryInstanceFreeableMemory	PrimaryInstanceFreeableMemory	Histogram	ClusterId , ClusterName
PrimaryInstanceFreeableMemory	PrimaryInstanceFreeableMemoryByShard	Histogram	ClusterId , ClusterName , ShardId
ReadThroughput	ReadThroughput	Histogram	ClusterId , ClusterName
ReadThroughput	ReadThroughputByShard	Histogram	ClusterId , ClusterName , ShardId
TTLDeletedDocuments	TTLDeletedDocuments	Sum	ClusterId , ClusterName
TTLDeletedDocuments	TTLDeletedDocumentsByShard	Sum	ClusterId , ClusterName , ShardId
VolumeBytesUsed	VolumeBytesUsed	Histogram	ClusterId , ClusterName
VolumeBytesUsed	VolumeBytesUsedByShard	Histogram	ClusterId , ClusterName , ShardId
VolumeReadIOPs	VolumeReadIOPs	Histogram	ClusterId , ClusterName
VolumeReadIOPs	VolumeReadIOPsByShard	Histogram	ClusterId , ClusterName , ShardId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
VolumeWriteIOPs	VolumeWriteIOPs	Histogram	ClusterId , ClusterName
VolumeWriteIOPs	VolumeWriteIOPsByShard	Histogram	ClusterId , ClusterName , ShardId
WriteThroughput	WriteThroughput	Histogram	ClusterId , ClusterName
WriteThroughput	WriteThroughputByShard	Histogram	ClusterId , ClusterName , ShardId

AWS/DynamoDB

The AWS/DynamoDB namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/dynamodb. All enriched instruments use Delta temporality.

- AWS::DynamoDB (service-level)
- AWS::DynamoDB::GlobalTable
- AWS::DynamoDB::Table
- index
- stream

AWS::DynamoDB (service-level)

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
UserErrors	UserErrors	Sum	

AWS::DynamoDB::GlobalTable

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PendingReplicationCount	PendingReplicationCount	Histogram	ReceivingRegion, TableName
ReplicationLag	ReplicationLag	Histogram	ReceivingRegion, SourceRegion, TableName
ReplicationLatency	ReplicationLatency	Histogram	ReceivingRegion, TableName

AWS::DynamoDB::Table

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AgeOfOldestUnreplicatedRecord	AgeOfOldestUnreplicatedRecord	Histogram	DelegatedOperation, TableName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConditionalCheckFailedRequests	ConditionalCheckFailedRequests	Sum	TableName
ConsumedChangeDataCaptureUnits	ConsumedChangeDataCaptureUnits	Sum	DelegatedOperation, TableName
ConsumedReadCapacityUnits	ConsumedReadCapacityUnits	Sum	TableName
ConsumedWriteCapacityUnits	ConsumedWriteCapacityUnits	Sum	TableName
ProvisionedReadCapacityUnits	ProvisionedReadCapacityUnits	Histogram	TableName
ProvisionedWriteCapacityUnits	ProvisionedWriteCapacityUnits	Histogram	TableName
ReadThrottleEvents	ReadThrottleEvents	Sum	TableName
ReturnedItemCount	ReturnedItemCount	Histogram	Operation, TableName
SuccessfulRequestLatency	SuccessfulRequestLatency	Histogram	Operation, TableName
SystemErrors	SystemErrors	Sum	Operation, TableName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ThrottledPutRecordCount	ThrottledPutRecordCount	Sum	DelegatedOperation, TableName
ThrottledRequests	ThrottledRequests	Sum	Operation, TableName
TimeToLiveDeletedItemCount	TimeToLiveDeletedItemCount	Sum	TableName
TransactionConflict	TransactionConflict	Sum	TableName
WriteThrottleEvents	WriteThrottleEvents	Sum	TableName

index

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OnlineIndexConsumedWriteCapacity	OnlineIndexConsumedWriteCapacity	Sum	GlobalSecondaryIndexName, TableName
OnlineIndexPercentageProgress	OnlineIndexPercentageProgress	Histogram	GlobalSecondaryInd

Metric	OpenTelemetry metric	Instrument type	Data point attributes
			exName , TableName
OnlineIndexThrottleEvents	OnlineIndexThrottleEvents	Sum	GlobalSecondaryIndexName , TableName

stream

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReturnedBytes	ReturnedBytes	Histogram	Operation , StreamLabel , TableName
ReturnedRecordsCount	ReturnedRecordsCount	Histogram	Operation , StreamLabel , TableName

AWS/EBS

The AWS/EBS namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/ec2. All enriched instruments use Delta temporality.

- AWS::EC2::Volume
- snapshot

AWS::EC2::Volume

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BurstBalance	BurstBalance	Histogram	VolumeId
VolumeAvgIOPS	VolumeAvgIOPS	Histogram	InstanceId , VolumeId
VolumeAvg ReadLatency	VolumeAvg ReadLatency	Histogram	InstanceId , VolumeId
VolumeAvg Throughput	VolumeAvg Throughput	Histogram	InstanceId , VolumeId
VolumeAvg WriteLatency	VolumeAvg WriteLatency	Histogram	InstanceId , VolumeId
VolumeCon sumedRead WriteOps	VolumeCon sumedRead WriteOps	Sum	VolumeId
VolumeIOP SExceededCheck	VolumeIOP SExceededCheck	Histogram	InstanceId , VolumeId
VolumeIdleTime	VolumeIdleTime	Sum	VolumeId
VolumeQue ueLength	VolumeQue ueLength	Histogram	VolumeId
VolumeReadBytes	VolumeReadBytes	Sum	VolumeId
VolumeReadOps	VolumeReadOps	Sum	VolumeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
VolumeStalledIOCheck	VolumeStalledIOCheck	Histogram	InstanceId , VolumeId
VolumeThroughputExceededCheck	VolumeThroughputExceededCheck	Histogram	InstanceId , VolumeId
VolumeThroughputPercentage	VolumeThroughputPercentage	Histogram	VolumeId
VolumeTotalReadTime	VolumeTotalReadTime	Sum	VolumeId
VolumeTotalWriteTime	VolumeTotalWriteTime	Sum	VolumeId
VolumeWriteBytes	VolumeWriteBytes	Sum	VolumeId
VolumeWriteOps	VolumeWriteOps	Sum	VolumeId

snapshot

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FastSnapshotRestor	FastSnapshotRestor	Histogram	AvailabilityZone , SnapshotId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
eCreditsBalance	eCreditsBalance		
FastSnapshotRestoreCreditsBucketSize	FastSnapshotRestoreCreditsBucketSize	Histogram	AvailabilityZone, SnapshotId

AWS/EC2

The AWS/EC2 namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/ec2. All enriched instruments use Delta temporality.

- AWS::EC2::Host
- AWS::EC2::Instance
- AWS::EC2::VPC
- image

AWS::EC2::Host

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DedicatedHostCPUUtilization	DedicatedHostCPUUtilization	Histogram	HostId

AWS::EC2::Instance

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`CPUCredit Balance`	`CPUCredit Balance`	Histogram	• `AutoScali ngGroupNa me` , `ImageId`, `InstanceId` , `InstanceType` • `AutoScali ngGroupName` , `InstanceId` • `ImageId`, `InstanceId` , `InstanceType` • `InstanceId`
`CPUCreditUsage`	`CPUCreditUsage`	Histogram	• `AutoScali ngGroupNa me` , `ImageId`, `InstanceId` , `InstanceType` • `AutoScali ngGroupName` , `InstanceId` • `ImageId`, `InstanceId` , `InstanceType` • `InstanceId`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUSurplusCreditBalance	CPUSurplusCreditBalance	Histogram	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
CPUSurplusCreditsCharged	CPUSurplusCreditsCharged	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUUtilization	CPUUtilization	Histogram	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
DiskReadBytes	DiskReadBytes	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DiskReadOps	DiskReadOps	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
DiskWriteBytes	DiskWriteBytes	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DiskWriteOps	DiskWriteOps	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
EBSByteBalance%	EBSByteBalance%	Histogram	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EBSIOBalance%	EBSIOBalance%	Histogram	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
EBSReadBytes	EBSReadBytes	Sum	InstanceId
EBSReadOps	EBSReadOps	Sum	InstanceId
EBSWriteBytes	EBSWriteBytes	Sum	InstanceId
EBSWriteOps	EBSWriteOps	Sum	InstanceId
GPUPowerUtilization	GPUPowerUtilization	Histogram	InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InstanceEBSIOPSExceededCheck	InstanceEBSIOPSExceededCheck	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
InstanceEBSThroughputExceededCheck	InstanceEBSThroughputExceededCheck	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MetadataNoToken	MetadataNoToken	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
MetadataNoTokenRejected	MetadataNoTokenRejected	Sum	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkIn	NetworkIn	Sum	• AutoScalingGroupName, ImageId, InstanceId, InstanceType • AutoScalingGroupName, InstanceId • ImageId, InstanceId, InstanceType • InstanceId
NetworkMirrorIn	NetworkMirrorIn	Sum	InstanceId
NetworkMirrorOut	NetworkMirrorOut	Sum	InstanceId
NetworkOut	NetworkOut	Sum	• AutoScalingGroupName, ImageId, InstanceId, InstanceType • AutoScalingGroupName, InstanceId • ImageId, InstanceId, InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkPackets In	NetworkPackets In	Sum	• AutoScalingGroupName , ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
NetworkPackets MirrorIn	NetworkPackets MirrorIn	Sum	InstanceId
NetworkPackets MirrorOut	NetworkPackets MirrorOut	Sum	InstanceId
NetworkPackets Out	NetworkPackets Out	Sum	• AutoScalingGroupName , ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkPacketsSkipMirrorIn	NetworkPacketsSkipMirrorIn	Sum	InstanceId
NetworkPacketsSkipMirrorOut	NetworkPacketsSkipMirrorOut	Sum	InstanceId
NetworkSkipMirrorIn	NetworkSkipMirrorIn	Sum	InstanceId
NetworkSkipMirrorOut	NetworkSkipMirrorOut	Sum	InstanceId
StatusCheckFailed	StatusCheckFailed	Histogram	• AutoScalingGroupName ,ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
StatusCheckFailed_Application	StatusCheckFailed_Application	Histogram	InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StatusCheckFailed_AttachedEBS	StatusCheckFailed_AttachedEBS	Histogram	• AutoScalingGroupName , ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId
StatusCheckFailed_Instance	StatusCheckFailed_Instance	Histogram	• AutoScalingGroupName , ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StatusCheckFailed_System	StatusCheckFailed_System	Histogram	• AutoScalingGroupName , ImageId, InstanceId , InstanceType • AutoScalingGroupName , InstanceId • ImageId, InstanceId , InstanceType • InstanceId

AWS::EC2::VPC

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkAddressUsage	NetworkAddressUsage	Histogram	Per-VPC Metrics
NetworkAddressUsageePeered	NetworkAddressUsageePeered	Histogram	Per-VPC Metrics

image

- cloud.resource_id enrichment: Supported

- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`CPUUtilization`	`CPUUtilization`	Histogram	`ImageId`
`DiskReadBytes`	`DiskReadBytes`	Sum	`ImageId`
`DiskReadOps`	`DiskReadOps`	Sum	`ImageId`
`DiskWriteBytes`	`DiskWriteBytes`	Sum	`ImageId`
`DiskWriteOps`	`DiskWriteOps`	Sum	`ImageId`
`NetworkIn`	`NetworkIn`	Sum	`ImageId`
`NetworkOut`	`NetworkOut`	Sum	`ImageId`
`NumberOfAvailableFastLaunchSnapshots`	`NumberOfAvailableFastLaunchSnapshots`	Histogram	`ImageId`, `ImageName`

AWS/EC2CapacityReservations

The `AWS/EC2CapacityReservations` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/ec2`. All enriched instruments use Delta temporality.

- AWS::EC2::CapacityReservation

AWS::EC2::CapacityReservation

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AvailableInstanceCount	AvailableInstanceCount	Histogram	CapacityReservationId
InstanceUtilization	InstanceUtilization	Histogram	CapacityReservationId
TotalInstanceCount	TotalInstanceCount	Histogram	CapacityReservationId
UsedInstanceCount	UsedInstanceCount	Histogram	CapacityReservationId

AWS/EFS

The AWS/EFS namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticfilesystem. All enriched instruments use Delta temporality.

- AWS::EFS::FileSystem

AWS::EFS::FileSystem

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BurstCreditBalance	BurstCreditBalance	Histogram	FileSystemId
ClientConnections	ClientConnections	Histogram	FileSystemId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DataReadIOBytes	DataReadIOBytes	Sum	FileSystemId
DataWriteIOBytes	DataWriteIOBytes	Sum	FileSystemId
MetadataIOBytes	MetadataIOBytes	Sum	FileSystemId
MeteredIOBytes	MeteredIOBytes	Sum	FileSystemId
PercentIOLimit	PercentIOLimit	Histogram	FileSystemId
PermittedThroughput	PermittedThroughput	Histogram	FileSystemId
StorageBytes	StorageBytes	Histogram	FileSystemId , StorageClass
TimeSinceLastSync	TimeSinceLastSync	Histogram	DestinationFileSystemId , FileSystemId
TotalIOBytes	TotalIOBytes	Sum	FileSystemId

AWS/EKS

The AWS/EKS namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/eks`. All enriched instruments use Delta temporality.

- AWS::EKS::Cluster

AWS::EKS::Cluster

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
apiserver_admissio n_webhook _rejectio n_count_ADMIT	apiserver_admissio n_webhook _rejectio n_count_ADMIT	Sum	ClusterName
apiserver_admissio n_webhook _rejectio n_count_V ALIDATING	apiserver_admissio n_webhook _rejectio n_count_V ALIDATING	Sum	ClusterName
apiserver_admissio n_webhook _request_total	apiserver_admissio n_webhook _request_total	Sum	ClusterName
apiserver_admissio n_webhook _request_ total_ADMIT	apiserver_admissio n_webhook _request_ total_ADMIT	Sum	ClusterName
apiserver_admissio n_webhook _request_ total_VAL IDATING	apiserver_admissio n_webhook _request_ total_VAL IDATING	Sum	ClusterName
apiserver_current_ inflight_	apiserver_current_ inflight_	Histogram	ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
requests_MUTATING	requests_MUTATING		
apiserver_current_inflight_requests_READONLY	apiserver_current_inflight_requests_READONLY	Histogram	ClusterName
apiserver_flowcontrol_current_executing_seats	apiserver_flowcontrol_current_executing_seats	Histogram	ClusterName
apiserver_request_duration_seconds_DELETE_P99	apiserver_request_duration_seconds_DELETE_P99	Histogram	ClusterName
apiserver_request_duration_seconds_GET_P99	apiserver_request_duration_seconds_GET_P99	Histogram	ClusterName
apiserver_request_duration_seconds_LIST_P99	apiserver_request_duration_seconds_LIST_P99	Histogram	ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
apiserver_request_duration_seconds_PATCH_P99	apiserver_request_duration_seconds_PATCH_P99	Histogram	ClusterName
apiserver_request_duration_seconds_POST_P99	apiserver_request_duration_seconds_POST_P99	Histogram	ClusterName
apiserver_request_duration_seconds_PUT_P99	apiserver_request_duration_seconds_PUT_P99	Histogram	ClusterName
apiserver_request_total	apiserver_request_total	Sum	ClusterName
apiserver_request_total_429	apiserver_request_total_429	Sum	ClusterName
apiserver_request_total_4XX	apiserver_request_total_4XX	Sum	ClusterName
apiserver_request_total_5XX	apiserver_request_total_5XX	Sum	ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
apiserver_request_total_LISTPODS	apiserver_request_total_LISTPODS	Sum	ClusterName
apiserver_storage_size_bytes	apiserver_storage_size_bytes	Histogram	ClusterName
etcd_mvcc_db_total_size_in_use_in_bytes	etcd_mvcc_db_total_size_in_use_in_bytes	Histogram	ClusterName
scheduler_pending_pods	scheduler_pending_pods	Histogram	ClusterName
scheduler_pending_pods_ACTIVEQ	scheduler_pending_pods_ACTIVEQ	Histogram	ClusterName
scheduler_pending_pods_BACKOFF	scheduler_pending_pods_BACKOFF	Histogram	ClusterName
scheduler_pending_pods_GATED	scheduler_pending_pods_GATED	Histogram	ClusterName
scheduler_pending_pods_UNSCHEDULABLE	scheduler_pending_pods_UNSCHEDULABLE	Histogram	ClusterName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
scheduler_schedule_attempts_ERROR	scheduler_schedule_attempts_ERROR	Sum	ClusterName
scheduler_schedule_attempts_SCHEDULED	scheduler_schedule_attempts_SCHEDULED	Sum	ClusterName
scheduler_schedule_attempts_UNSCHEDULABLE	scheduler_schedule_attempts_UNSCHEDULABLE	Sum	ClusterName
scheduler_schedule_attempts_total	scheduler_schedule_attempts_total	Sum	ClusterName

AWS/ELB

The AWS/ELB namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/elasticloadbalancing`. All enriched instruments use Delta temporality.

- AWS::ElasticLoadBalancing::LoadBalancer

AWS::ElasticLoadBalancing::LoadBalancer

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BackendConnectionErrors	BackendConnectionErrors	Sum	AvailabilityZone, LoadBalancerName
DesyncMitigationMode_NonCompliant_Request_Count	DesyncMitigationMode_NonCompliant_Request_Count	Sum	AvailabilityZone, LoadBalancerName
EstimatedALBActiveConnectionCount	EstimatedALBActiveConnectionCount	Histogram	LoadBalancerName
EstimatedALBNewConnectionCount	EstimatedALBNewConnectionCount	Sum	LoadBalancerName
EstimatedProcessedBytes	EstimatedProcessedBytes	Sum	LoadBalancerName
HTTPCode_Backend_2XX	HTTPCode_Backend_2XX	Sum	AvailabilityZone, LoadBalancerName
HTTPCode_Backend_3XX	HTTPCode_Backend_3XX	Sum	AvailabilityZone, LoadBalancerName
HTTPCode_Backend_4XX	HTTPCode_Backend_4XX	Sum	AvailabilityZone,

Metric	OpenTelemetry metric	Instrument type	Data point attributes
			LoadBalancerName
HTTPCode_Backend_5XX	HTTPCode_Backend_5XX	Sum	AvailabilityZone , LoadBalancerName
HTTPCode_ELB_4XX	HTTPCode_ELB_4XX	Sum	AvailabilityZone , LoadBalancerName
HTTPCode_ELB_5XX	HTTPCode_ELB_5XX	Sum	AvailabilityZone , LoadBalancerName
HealthyHostCount	HealthyHostCount	Histogram	AvailabilityZone , LoadBalancerName
Latency	Latency	Histogram	AvailabilityZone , LoadBalancerName
RequestCount	RequestCount	Sum	AvailabilityZone , LoadBalancerName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SpilloverCount	SpilloverCount	Sum	AvailabilityZone , LoadBalancerName
SurgeQueueLength	SurgeQueueLength	Histogram	AvailabilityZone , LoadBalancerName
UnHealthyHostCount	UnHealthyHostCount	Histogram	AvailabilityZone , LoadBalancerName

AWS/EMRServerless

The AWS/EMRServerless namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/emr-serverless. All enriched instruments use Delta temporality.

- AWS::EMRServerless::Application

AWS::EMRServerless::Application

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUAllocated	CPUAllocated	Histogram	ApplicationId , ApplicationName , CapacityA

Metric	OpenTelemetry metric	Instrument type	Data point attributes
			llocationType , WorkerType
CancelledJobs	CancelledJobs	Sum	ApplicationId , ApplicationName
CancellingJobs	CancellingJobs	Histogram	ApplicationId , ApplicationName
FailedJobs	FailedJobs	Sum	ApplicationId , ApplicationName
IdleWorkerCount	IdleWorkerCount	Histogram	ApplicationId , ApplicationName , CapacityA llocationType , WorkerType
MaxCPUAllowed	MaxCPUAllowed	Histogram	ApplicationId , ApplicationName
MaxMemory Allowed	MaxMemory Allowed	Histogram	ApplicationId , ApplicationName
MaxStorag eAllowed	MaxStorag eAllowed	Histogram	ApplicationId , ApplicationName
MemoryAllocated	MemoryAllocated	Histogram	ApplicationId , ApplicationName , CapacityA llocationType , WorkerType

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PendingCreationWorkerCount	PendingCreationWorkerCount	Histogram	ApplicationId , ApplicationName , CapacityAllocationType , WorkerType
PendingJobs	PendingJobs	Histogram	ApplicationId , ApplicationName
RunningJobs	RunningJobs	Histogram	ApplicationId , ApplicationName
RunningWorkerCount	RunningWorkerCount	Histogram	ApplicationId , ApplicationName , CapacityAllocationType , WorkerType
ScheduledJobs	ScheduledJobs	Histogram	ApplicationId , ApplicationName
StorageAllocated	StorageAllocated	Histogram	ApplicationId , ApplicationName , CapacityAllocationType , WorkerType
SubmittedJobs	SubmittedJobs	Sum	ApplicationId , ApplicationName
SuccessJobs	SuccessJobs	Sum	ApplicationId , ApplicationName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalWorkerCount	TotalWorkerCount	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, WorkerType
WorkerCpuAllocated	WorkerCpuAllocated	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerCpuUsed	WorkerCpuUsed	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerEphemeralStorageAllocated	WorkerEphemeralStorageAllocated	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerEphemeralStorageUsed	WorkerEphemeralStorageUsed	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WorkerMemoryAllocated	WorkerMemoryAllocated	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerMemoryUsed	WorkerMemoryUsed	Histogram	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerStorageReadBytes	WorkerStorageReadBytes	Sum	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType
WorkerStorageWriteBytes	WorkerStorageWriteBytes	Sum	ApplicationId, ApplicationName, CapacityAllocationType, JobId, JobName, WorkerType

AWS/ES

The AWS/ES namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/es. All enriched instruments use Delta temporality.

- AWS::OpenSearchService::Domain

AWS::OpenSearchService::Domain

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`2xx`	`2xx`	Sum	`ClientId`, `DomainName`
`3xx`	`3xx`	Sum	`ClientId`, `DomainName`
`4xx`	`4xx`	Sum	`ClientId`, `DomainName`
`5xx`	`5xx`	Sum	`ClientId`, `DomainName`
`ADAnomaly Detectors IndexStatus.red`	`ADAnomaly Detectors IndexStatus.red`	Histogram	`ClientId`, `DomainName`
`ADAnomaly Detectors IndexStatusIndexExists`	`ADAnomaly Detectors IndexStatusIndexExists`	Histogram	`ClientId`, `DomainName`
`ADAnomaly ResultsIndexStatus.red`	`ADAnomaly ResultsIndexStatus.red`	Histogram	`ClientId`, `DomainName`
`ADAnomaly ResultsIn`	`ADAnomaly ResultsIn`	Histogram	`ClientId`, `DomainName`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
dexStatus IndexExists	dexStatus IndexExists		
ADExecute FailureCount	ADExecute FailureCount	Sum	ClientId, DomainName , NodeId
ADExecute RequestCount	ADExecute RequestCount	Sum	ClientId, DomainName , NodeId
ADHCExecu teFailureCount	ADHCExecu teFailureCount	Sum	ClientId, DomainName , NodeId
ADHCExecu teRequestCount	ADHCExecu teRequestCount	Sum	ClientId, DomainName , NodeId
ADModelsC heckpoint IndexStat us.red	ADModelsC heckpoint IndexStat us.red	Histogram	ClientId, DomainName
ADModelsC heckpoint IndexStat usIndexExists	ADModelsC heckpoint IndexStat usIndexExists	Histogram	ClientId, DomainName
ADPluginU nhealthy	ADPluginU nhealthy	Histogram	ClientId, DomainName
ActiveDataNode	ActiveDataNode	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AlertingDegraded	AlertingDegraded	Histogram	ClientId, DomainName
AlertingIndexExists	AlertingIndexExists	Histogram	ClientId, DomainName
AlertingIndexStatus.green	AlertingIndexStatus.green	Histogram	ClientId, DomainName
AlertingIndexStatus.red	AlertingIndexStatus.red	Histogram	ClientId, DomainName
AlertingIndexStatus.yellow	AlertingIndexStatus.yellow	Histogram	ClientId, DomainName
AlertingNodesNotOnSchedule	AlertingNodesNotOnSchedule	Histogram	ClientId, DomainName
AlertingNodesOnSchedule	AlertingNodesOnSchedule	Histogram	ClientId, DomainName
AlertingScheduledJobEnabled	AlertingScheduledJobEnabled	Histogram	ClientId, DomainName
AsyncQueryCancelApiFailedRequestCusErrCount	AsyncQueryCancelApiFailedRequestCusErrCount	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AsyncQueryCancelApiFailedRequestSysErrCount	AsyncQueryCancelApiFailedRequestSysErrCount	Sum	ClientId, DomainName
AsyncQueryCancelApiRequestCount	AsyncQueryCancelApiRequestCount	Sum	ClientId, DomainName
AsyncQueryCreateApiFailedRequestCusErrCount	AsyncQueryCreateApiFailedRequestCusErrCount	Sum	ClientId, DomainName
AsyncQueryCreateApiFailedRequestSysErrCount	AsyncQueryCreateApiFailedRequestSysErrCount	Sum	ClientId, DomainName
AsyncQueryCreateApiRequestCount	AsyncQueryCreateApiRequestCount	Sum	ClientId, DomainName
AsyncQueryGetApiFailedRequestCusErrCount	AsyncQueryGetApiFailedRequestCusErrCount	Sum	ClientId, DomainName
AsyncQueryGetApiFailedRequestSysErrCount	AsyncQueryGetApiFailedRequestSysErrCount	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AsyncQueryGetApiRequestCount	AsyncQueryGetApiRequestCount	Sum	ClientId, DomainName
AsynchronousSearchCancelled	AsynchronousSearchCancelled	Sum	ClientId, DomainName , NodeId
AsynchronousSearchCompletionRate	AsynchronousSearchCompletionRate	Sum	ClientId, DomainName , NodeId
AsynchronousSearchFailureRate	AsynchronousSearchFailureRate	Sum	ClientId, DomainName , NodeId
AsynchronousSearchInitializedRate	AsynchronousSearchInitializedRate	Sum	ClientId, DomainName , NodeId
AsynchronousSearchPersistFailedRate	AsynchronousSearchPersistFailedRate	Sum	ClientId, DomainName , NodeId
AsynchronousSearchPersistRate	AsynchronousSearchPersistRate	Sum	ClientId, DomainName , NodeId
AsynchronousSearchRejected	AsynchronousSearchRejected	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AsynchronousSearchRunningCurrent	AsynchronousSearchRunningCurrent	Histogram	ClientId, DomainName , NodeId
AsynchronousSearchStoreHealthRed	AsynchronousSearchStoreHealthRed	Histogram	ClientId, DomainName
AsynchronousSearchStoreSize	AsynchronousSearchStoreSize	Histogram	ClientId, DomainName
AsynchronousSearchStoredResponseCount	AsynchronousSearchStoredResponseCount	Histogram	ClientId, DomainName
AsynchronousSearchSubmissionRate	AsynchronousSearchSubmissionRate	Sum	ClientId, DomainName , NodeId
AutoFollowLeaderCallFailure	AutoFollowLeaderCallFailure	Histogram	ClientId, DomainName , LeaderAlias
AutoFollowNumFailedStartReplication	AutoFollowNumFailedStartReplication	Sum	ClientId, DomainName , LeaderAlias
AutoFollowNumSuccessStartReplication	AutoFollowNumSuccessStartReplication	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AutoTuneChangesHistoryHeapSize	AutoTuneChangesHistoryHeapSize	Histogram	AutotuningType ,ClientId, DomainName , TargetId
AutoTuneFailed	AutoTuneFailed	Sum	AutotuningType ,ClientId, DomainName , TargetId
AutoTuneSucceeded	AutoTuneSucceeded	Sum	AutotuningType ,ClientId, DomainName , TargetId
AutoTuneValue	AutoTuneValue	Histogram	AutotuningType ,ClientId, DomainName , TargetId
AutomatedSnapshotFailure	AutomatedSnapshotFailure	Histogram	ClientId, DomainName
AvgPointInTimeAliveTime	AvgPointInTimeAliveTime	Histogram	ClientId, DomainName
BurstBalance	BurstBalance	Histogram	ClientId, DomainName , NodeId
CPUCreditBalance	CPUCreditBalance	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUUtilization	CPUUtilization	Histogram	ClientId, DomainName , NodeId
CanMatchCurrent	CanMatchCurrent	Histogram	ClientId, DomainName , NodeId
CanMatchTimeInMillis	CanMatchTimeInMillis	Histogram	ClientId, DomainName , NodeId
CanMatchTotal	CanMatchTotal	Sum	ClientId, DomainName , NodeId
ClusterIndexWritesBlocked	ClusterIndexWritesBlocked	Histogram	ClientId, DomainName
ClusterStatus.green	ClusterStatus.green	Histogram	ClientId, DomainName
ClusterStatus.red	ClusterStatus.red	Histogram	ClientId, DomainName
ClusterStatus.yellow	ClusterStatus.yellow	Histogram	ClientId, DomainName
ClusterUsedSpace	ClusterUsedSpace	Histogram	ClientId, DomainName
ColdStorageSpaceUtilization	ColdStorageSpaceUtilization	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ColdToWarmMigrationFailureCount	ColdToWarmMigrationFailureCount	Sum	ClientId, DomainName
ColdToWarmMigrationLatency	ColdToWarmMigrationLatency	Histogram	ClientId, DomainName
ColdToWarmMigrationQueueSize	ColdToWarmMigrationQueueSize	Histogram	ClientId, DomainName
ColdToWarmMigrationSuccessCount	ColdToWarmMigrationSuccessCount	Sum	ClientId, DomainName
ComponentTemplateCount	ComponentTemplateCount	Histogram	ClientId, DomainName
ConcurrentSearchLatency	ConcurrentSearchLatency	Histogram	ClientId, DomainName , NodeId
ConcurrentSearchRate	ConcurrentSearchRate	Sum	ClientId, DomainName , NodeId
CoordinatingWriteRejected	CoordinatingWriteRejected	Sum	ClientId, DomainName , NodeId
CoordinatorCPUUtilization	CoordinatorCPUUtilization	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CoordinatorFreeStorageSpace	CoordinatorFreeStorageSpace	Histogram	ClientId, DomainName , NodeId
CoordinatorJVMGCOldCollectionCount	CoordinatorJVMGCOldCollectionCount	Sum	ClientId, DomainName , NodeId
CoordinatorJVMGCOldCollectionTime	CoordinatorJVMGCOldCollectionTime	Sum	ClientId, DomainName , NodeId
CoordinatorJVMGCYoungCollectionCount	CoordinatorJVMGCYoungCollectionCount	Sum	ClientId, DomainName , NodeId
CoordinatorJVMGCYoungCollectionTime	CoordinatorJVMGCYoungCollectionTime	Sum	ClientId, DomainName , NodeId
CoordinatorJVMMemoryPressure	CoordinatorJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
CoordinatorOldGenJVMMemoryPressure	CoordinatorOldGenJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Coordinat orSysMemo ryUtilization	Coordinat orSysMemo ryUtilization	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolForce_ mergeQueue	Coordinat orThreadp oolForce_ mergeQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolForce_ mergeRejected	Coordinat orThreadp oolForce_ mergeRejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolForce_ mergeThreads	Coordinat orThreadp oolForce_ mergeThreads	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolIndexQueue	Coordinat orThreadp oolIndexQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolIndexR ejected	Coordinat orThreadp oolIndexR ejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolIndexT hreads	Coordinat orThreadp oolIndexT hreads	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Coordinat orThreadp oolOpendi stro_moni tor_runne rQueue	Coordinat orThreadp oolOpendi stro_moni tor_runne rQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolOpendi stro_moni tor_runne rRejected	Coordinat orThreadp oolOpendi stro_moni tor_runne rRejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolOpendi stro_moni tor_runne rThreads	Coordinat orThreadp oolOpendi stro_moni tor_runne rThreads	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolSearchQueue	Coordinat orThreadp oolSearchQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolSearch Rejected	Coordinat orThreadp oolSearch Rejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolSearch Threads	Coordinat orThreadp oolSearch Threads	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Coordinat orThreadp oolWriteQueue	Coordinat orThreadp oolWriteQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolWriteR ejected	Coordinat orThreadp oolWriteR ejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolWriteT hreads	Coordinat orThreadp oolWriteT hreads	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolsql-wo rkerQueue	Coordinat orThreadp oolsql-wo rkerQueue	Histogram	ClientId, DomainName , NodeId
Coordinat orThreadp oolsql-wo rkerRejected	Coordinat orThreadp oolsql-wo rkerRejected	Sum	ClientId, DomainName , NodeId
Coordinat orThreadp oolsql-wo rkerThreads	Coordinat orThreadp oolsql-wo rkerThreads	Histogram	ClientId, DomainName , NodeId
CrossClus terInboun dReplicat ionRequests	CrossClus terInboun dReplicat ionRequests	Sum	ClientId, ConnectionId , DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CrossClusterInboundRequests	CrossClusterInboundRequests	Sum	ClientId, ConnectionId , DomainName
CrossClusterInboundSearchRequests	CrossClusterInboundSearchRequests	Sum	ClientId, ConnectionId , DomainName
CrossClusterOutboundConnections	CrossClusterOutboundConnections	Histogram	ClientId, ConnectionId , DomainName
CrossClusterOutboundReplicationRequests	CrossClusterOutboundReplicationRequests	Sum	ClientId, ConnectionId , DomainName
CrossClusterOutboundRequests	CrossClusterOutboundRequests	Sum	ClientId, ConnectionId , DomainName
CrossClusterOutboundSearchRequests	CrossClusterOutboundSearchRequests	Sum	ClientId, ConnectionId , DomainName
CurrentPointInTime	CurrentPointInTime	Histogram	ClientId, DomainName , NodeId
DataNodes	DataNodes	Histogram	ActiveAZ, ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DataNodes Shards.active	DataNodes Shards.active	Histogram	ActiveAZ, ClientId, DomainName
DataNodes Shards.in itialising	DataNodes Shards.in itialising	Histogram	ActiveAZ, ClientId, DomainName
DataNodes Shards.re locating	DataNodes Shards.re locating	Histogram	ActiveAZ, ClientId, DomainName
DataNodes Shards.un assigned	DataNodes Shards.un assigned	Histogram	ActiveAZ, ClientId, DomainName
DeletedDo cuments	DeletedDo cuments	Histogram	ClientId, DomainName
DfsPreQue ryCurrent	DfsPreQue ryCurrent	Histogram	ClientId, DomainName , NodeId
DfsPreQue ryTimeInMillis	DfsPreQue ryTimeInMillis	Histogram	ClientId, DomainName , NodeId
DfsPreQue ryTotal	DfsPreQue ryTotal	Sum	ClientId, DomainName , NodeId
DfsQueryCurrent	DfsQueryCurrent	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DfsQueryTimeInMillis	DfsQueryTimeInMillis	Histogram	ClientId, DomainName , NodeId
DfsQueryTotal	DfsQueryTotal	Sum	ClientId, DomainName , NodeId
DiskQueueDepth	DiskQueueDepth	Histogram	ClientId, DomainName
ESReportingFailedRequestSysErrCount	ESReportingFailedRequestSysErrCount	Sum	ClientId, DomainName
ESReportingFailedRequestUserErrCount	ESReportingFailedRequestUserErrCount	Sum	ClientId, DomainName
ESReportingRequestCount	ESReportingRequestCount	Sum	ClientId, DomainName
ESReportingSuccessCount	ESReportingSuccessCount	Sum	ClientId, DomainName
ElasticsearchRequests	ElasticsearchRequests	Sum	ClientId, DomainName
EnforcedWorkloadGroupCount	EnforcedWorkloadGroupCount	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ExpandCurrent	ExpandCurrent	Histogram	ClientId, DomainName , NodeId
ExpandTimeInMillis	ExpandTimeInMillis	Histogram	ClientId, DomainName , NodeId
ExpandTotal	ExpandTotal	Sum	ClientId, DomainName , NodeId
FetchCurrent	FetchCurrent	Histogram	ClientId, DomainName , NodeId
FetchTimeInMillis	FetchTimeInMillis	Histogram	ClientId, DomainName , NodeId
FetchTotal	FetchTotal	Sum	ClientId, DomainName , NodeId
FollowerCheckPoint	FollowerCheckPoint	Histogram	ClientId, ConnectionId , DomainName
ForecastCheckpointIndexStatus.red	ForecastCheckpointIndexStatus.red	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ForecastCheckpointIndexStatusIndexExists	ForecastCheckpointIndexStatusIndexExists	Histogram	ClientId, DomainName
ForecastConfigIndexStatus.red	ForecastConfigIndexStatus.red	Histogram	ClientId, DomainName
ForecastConfigIndexStatusIndexExists	ForecastConfigIndexStatusIndexExists	Histogram	ClientId, DomainName
ForecastExecuteFailureCount	ForecastExecuteFailureCount	Sum	ClientId, DomainName , NodeId
ForecastExecuteRequestCount	ForecastExecuteRequestCount	Sum	ClientId, DomainName , NodeId
ForecastHCExecuteFailureCount	ForecastHCExecuteFailureCount	Sum	ClientId, DomainName , NodeId
ForecastHCExecuteRequestCount	ForecastHCExecuteRequestCount	Sum	ClientId, DomainName , NodeId
ForecastModelCorruptionCount	ForecastModelCorruptionCount	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ForecastResultsIndexStatus.red	ForecastResultsIndexStatus.red	Histogram	ClientId, DomainName
ForecastResultsIndexStatusIndexExists	ForecastResultsIndexStatusIndexExists	Histogram	ClientId, DomainName
ForecastStateIndexStatus.red	ForecastStateIndexStatus.red	Histogram	ClientId, DomainName
ForecastStateIndexStatusIndexExists	ForecastStateIndexStatusIndexExists	Histogram	ClientId, DomainName
ForecastUnhealthy	ForecastUnhealthy	Histogram	ClientId, DomainName
ForecasterCount	ForecasterCount	Histogram	ClientId, DomainName
FreeStorageSpace	FreeStorageSpace	Histogram	ClientId, DomainName , NodeId
HCForecasterCount	HCForecasterCount	Histogram	ClientId, DomainName
HasActivePointInTime	HasActivePointInTime	Histogram	ClientId, DomainName
HasUsedPointInTime	HasUsedPointInTime	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HighSwapUsage	HighSwapUsage	Histogram	ClientId, DomainName , NodeId
HotStorageSpaceUtilization	HotStorageSpaceUtilization	Histogram	ClientId, DomainName
HotToWarmMigrationFailureCount	HotToWarmMigrationFailureCount	Sum	ClientId, DomainName
HotToWarmMigrationForceMergeLatency	HotToWarmMigrationForceMergeLatency	Histogram	ClientId, DomainName
HotToWarmMigrationProcessingLatency	HotToWarmMigrationProcessingLatency	Histogram	ClientId, DomainName
HotToWarmMigrationQueueSize	HotToWarmMigrationQueueSize	Histogram	ClientId, DomainName
HotToWarmMigrationSnapshotLatency	HotToWarmMigrationSnapshotLatency	Histogram	ClientId, DomainName
HotToWarmMigrationSuccessCount	HotToWarmMigrationSuccessCount	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HotToWarmMigrationSuccessLatency	HotToWarmMigrationSuccessLatency	Histogram	ClientId, DomainName
ISMPolicyCount	ISMPolicyCount	Histogram	ClientId, DomainName
InFlightFetches	InFlightFetches	Histogram	ClientId, DomainName
IndexingLatency	IndexingLatency	Histogram	ClientId, DomainName , NodeId
IndexingRate	IndexingRate	Sum	ClientId, DomainName , NodeId
InvalidHostHeaderRequests	InvalidHostHeaderRequests	Sum	ClientId, DomainName
IopsThrottle	IopsThrottle	Histogram	ClientId, DomainName , NodeId
JVMGCOldCollectionCount	JVMGCOldCollectionCount	Sum	ClientId, DomainName , NodeId
JVMGCOldCollectionTime	JVMGCOldCollectionTime	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
JVMGCYoungCollectionCount	JVMGCYoungCollectionCount	Sum	ClientId, DomainName , NodeId
JVMGCYoungCollectionTime	JVMGCYoungCollectionTime	Sum	ClientId, DomainName , NodeId
JVMMemoryPressure	JVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
KMSKeyError	KMSKeyError	Histogram	ClientId, DomainName
KMSKeyInaccessible	KMSKeyInaccessible	Histogram	ClientId, DomainName
KNNCacheCapacityReached	KNNCacheCapacityReached	Histogram	ClientId, DomainName , NodeId
KNNCircuitBreakerTriggered	KNNCircuitBreakerTriggered	Histogram	ClientId, DomainName
KNNEvictionCount	KNNEvictionCount	Sum	ClientId, DomainName , NodeId
KNNFaissInitialized	KNNFaissInitialized	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KNNGraphIndexErrors	KNNGraphIndexErrors	Sum	ClientId, DomainName , NodeId
KNNGraphIndexRequests	KNNGraphIndexRequests	Sum	ClientId, DomainName , NodeId
KNNGraphMemoryUsage	KNNGraphMemoryUsage	Histogram	ClientId, DomainName , NodeId
KNNGraphMemoryUsagePercentage	KNNGraphMemoryUsagePercentage	Histogram	ClientId, DomainName , NodeId
KNNGraphQueryErrors	KNNGraphQueryErrors	Sum	ClientId, DomainName , NodeId
KNNGraphQueryRequests	KNNGraphQueryRequests	Sum	ClientId, DomainName , NodeId
KNNHitCount	KNNHitCount	Sum	ClientId, DomainName , NodeId
KNNLoadExceptionCount	KNNLoadExceptionCount	Sum	ClientId, DomainName , NodeId
KNNLoadSuccessCount	KNNLoadSuccessCount	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KNNLucene Initialized	KNNLucene Initialized	Histogram	ClientId, DomainName , NodeId
KNNMissCount	KNNMissCount	Sum	ClientId, DomainName , NodeId
KNNNmslib Initialized	KNNNmslib Initialized	Histogram	ClientId, DomainName , NodeId
KNNQueryR equests	KNNQueryR equests	Sum	ClientId, DomainName , NodeId
KNNRemote BuildEnabled	KNNRemote BuildEnabled	Histogram	ClientId, DomainName
KNNRemote IndexBuil dFailureCount	KNNRemote IndexBuil dFailureCount	Sum	ClientId, DomainName
KNNRemote IndexBuil dSuccessCount	KNNRemote IndexBuil dSuccessCount	Sum	ClientId, DomainName
KNNScript Compilati onErrors	KNNScript Compilati onErrors	Sum	ClientId, DomainName , NodeId
KNNScript Compilations	KNNScript Compilations	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KNNScript QueryErrors	KNNScript QueryErrors	Sum	ClientId, DomainName , NodeId
KNNScript QueryRequests	KNNScript QueryRequests	Sum	ClientId, DomainName , NodeId
KNNTotalL oadTime	KNNTotalL oadTime	Sum	ClientId, DomainName , NodeId
KNNTraini ngErrors	KNNTraini ngErrors	Sum	ClientId, DomainName , NodeId
KNNTraini ngMemoryUsage	KNNTraini ngMemoryUsage	Histogram	ClientId, DomainName , NodeId
KNNTraini ngMemoryU sagePercentage	KNNTraini ngMemoryU sagePercentage	Histogram	ClientId, DomainName , NodeId
KNNTraini ngRequests	KNNTraini ngRequests	Sum	ClientId, DomainName , NodeId
KibanaCon currentCo nnections	KibanaCon currentCo nnections	Histogram	ClientId, DomainName , NodeId
KibanaHea lthyNode	KibanaHea lthyNode	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KibanaHealthyNodes	KibanaHealthyNodes	Histogram	ClientId, DomainName
KibanaHeapTotal	KibanaHeapTotal	Histogram	ClientId, DomainName , NodeId
KibanaHeapUsed	KibanaHeapUsed	Histogram	ClientId, DomainName , NodeId
KibanaHeapUtilization	KibanaHeapUtilization	Histogram	ClientId, DomainName , NodeId
KibanaIndexMigrationFailed	KibanaIndexMigrationFailed	Histogram	ClientId, DomainName
KibanaOS1MinuteLoad	KibanaOS1MinuteLoad	Histogram	ClientId, DomainName , NodeId
KibanaRequestTotal	KibanaRequestTotal	Sum	ClientId, DomainName , NodeId
KibanaResponseTimesMaxInMillis	KibanaResponseTimesMaxInMillis	Histogram	ClientId, DomainName , NodeId
LTRFeatureMemoryUsageInBytes	LTRFeatureMemoryUsageInBytes	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LTRFeaturesetMemoryUsageInBytes	LTRFeaturesetMemoryUsageInBytes	Histogram	ClientId, DomainName , NodeId
LTRModelMemoryUsageInBytes	LTRModelMemoryUsageInBytes	Histogram	ClientId, DomainName , NodeId
LTRPluginUnhealthy	LTRPluginUnhealthy	Histogram	ClientId, DomainName
LTRRequestErrorCount	LTRRequestErrorCount	Sum	ClientId, DomainName , NodeId
LTRRequestTotalCount	LTRRequestTotalCount	Sum	ClientId, DomainName , NodeId
LTRStatus.red	LTRStatus.red	Histogram	ClientId, DomainName
LeaderCheckPoint	LeaderCheckPoint	Histogram	ClientId, DomainName , LeaderAlias
MLCircuitBreakerTriggerCount	MLCircuitBreakerTriggerCount	Sum	ClientId, DomainName , NodeId
MLCommonsPluginUnhealthy	MLCommonsPluginUnhealthy	Histogram	ClientId, DomainName
MLConnectorCount	MLConnectorCount	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MLConnectorIndexStatus.red	MLConnectorIndexStatus.red	Histogram	ClientId, DomainName
MLConnectorIndexStatusIndexExists	MLConnectorIndexStatusIndexExists	Histogram	ClientId, DomainName
MLDeployedModelCount	MLDeployedModelCount	Histogram	ClientId, DomainName , NodeId
MLExecutingTaskCount	MLExecutingTaskCount	Histogram	ClientId, DomainName , NodeId
MLFailureCount	MLFailureCount	Sum	ClientId, DomainName , NodeId
MLModelCount	MLModelCount	Histogram	ClientId, DomainName
MLModelIndexStatus.red	MLModelIndexStatus.red	Histogram	ClientId, DomainName
MLModelIndexStatusIndexExists	MLModelIndexStatusIndexExists	Histogram	ClientId, DomainName
MLRequestCount	MLRequestCount	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MLTaskIndexStatus.red	MLTaskIndexStatus.red	Histogram	ClientId, DomainName
MLTaskIndexStatusIndexExists	MLTaskIndexStatusIndexExists	Histogram	ClientId, DomainName
MasterCPUCreditBalance	MasterCPUCreditBalance	Histogram	ClientId, DomainName
MasterCPUUtilization	MasterCPUUtilization	Histogram	ClientId, DomainName , NodeId
MasterFreeStorageSpace	MasterFreeStorageSpace	Histogram	ClientId, DomainName , NodeId
MasterJVMMemoryPressure	MasterJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
MasterOldGenJVMMemoryPressure	MasterOldGenJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
MasterReachableFromNode	MasterReachableFromNode	Histogram	ClientId, DomainName
MasterSysMemoryUtilization	MasterSysMemoryUtilization	Histogram	ClientId, DomainName , NodeId
MaxProvisionedThroughput	MaxProvisionedThroughput	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MlCircuitBreakerTriggerCount	MlCircuitBreakerTriggerCount	Sum	ClientId, DomainName , NodeId
MlConnectorCount	MlConnectorCount	Histogram	ClientId, DomainName
MlConnectorIndexStatus.red	MlConnectorIndexStatus.red	Histogram	ClientId, DomainName
MlConnectorIndexStatusIndexExists	MlConnectorIndexStatusIndexExists	Histogram	ClientId, DomainName
MlDeployedModelCount	MlDeployedModelCount	Histogram	ClientId, DomainName , NodeId
MlExecutingTaskCount	MlExecutingTaskCount	Histogram	ClientId, DomainName , NodeId
MlFailureCount	MlFailureCount	Sum	ClientId, DomainName , NodeId
MlModelCount	MlModelCount	Histogram	ClientId, DomainName
MlModelIndexStatus.red	MlModelIndexStatus.red	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MlModelIndexStatusIndexExists	MlModelIndexStatusIndexExists	Histogram	ClientId, DomainName
MlNodeExecutingTaskCount	MlNodeExecutingTaskCount	Histogram	ClientId, DomainName , NodeId
MlNodeTotalCircuitBreakerTriggerCount	MlNodeTotalCircuitBreakerTriggerCount	Sum	ClientId, DomainName , NodeId
MlNodeTotalFailureCount	MlNodeTotalFailureCount	Sum	ClientId, DomainName , NodeId
MlNodeTotalModelCount	MlNodeTotalModelCount	Histogram	ClientId, DomainName , NodeId
MlNodeTotalRequestCount	MlNodeTotalRequestCount	Sum	ClientId, DomainName , NodeId
MlRequestCount	MlRequestCount	Sum	ClientId, DomainName , NodeId
MlTaskIndexStatus.red	MlTaskIndexStatus.red	Histogram	ClientId, DomainName
MlTaskIndexStatusIndexExists	MlTaskIndexStatusIndexExists	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Nodes	Nodes	Histogram	ClientId, DomainName
OldGenJVM MemoryPressure	OldGenJVM MemoryPressure	Histogram	ClientId, DomainName , NodeId
OpenContexts	OpenContexts	Histogram	ClientId, DomainName , NodeId
OpenContexts-BeagleStone	OpenContexts-BeagleStone	Histogram	ClientId, DomainName
OpenSearchDashboardsConcurrentConnections	OpenSearchDashboardsConcurrentConnections	Histogram	ClientId, DomainName , NodeId
OpenSearchDashboardsHealthyNode	OpenSearchDashboardsHealthyNode	Histogram	ClientId, DomainName , NodeId
OpenSearchDashboardsHealthyNodes	OpenSearchDashboardsHealthyNodes	Histogram	ClientId, DomainName
OpenSearchDashboardsHeapTotal	OpenSearchDashboardsHeapTotal	Histogram	ClientId, DomainName , NodeId
OpenSearchDashboardsHeapUsed	OpenSearchDashboardsHeapUsed	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OpenSearchDashboardsHeapUtilization	OpenSearchDashboardsHeapUtilization	Histogram	ClientId, DomainName , NodeId
OpenSearchDashboardsIndexMigrationFailed	OpenSearchDashboardsIndexMigrationFailed	Histogram	ClientId, DomainName
OpenSearchDashboardsOS1MinuteLoad	OpenSearchDashboardsOS1MinuteLoad	Histogram	ClientId, DomainName , NodeId
OpenSearchDashboardsRequestTotal	OpenSearchDashboardsRequestTotal	Sum	ClientId, DomainName , NodeId
OpenSearchDashboardsResponseTimesMaxInMillis	OpenSearchDashboardsResponseTimesMaxInMillis	Histogram	ClientId, DomainName , NodeId
OpenSearchRequests	OpenSearchRequests	Sum	ClientId, DomainName
OpensearchDashboardsReportingFailedRequestSysErrCount	OpensearchDashboardsReportingFailedRequestSysErrCount	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OpensearchDashboardsReportingFailedRequestUserErrCount	OpensearchDashboardsReportingFailedRequestUserErrCount	Sum	ClientId, DomainName
OpensearchDashboardsReportingRequestCount	OpensearchDashboardsReportingRequestCount	Sum	ClientId, DomainName
OpensearchDashboardsReportingSuccessCount	OpensearchDashboardsReportingSuccessCount	Sum	ClientId, DomainName
PPLFailedRequestCountByCusErr	PPLFailedRequestCountByCusErr	Sum	ClientId, DomainName
PPLFailedRequestCountBySysErr	PPLFailedRequestCountBySysErr	Sum	ClientId, DomainName
PPLRequestCount	PPLRequestCount	Sum	ClientId, DomainName
PrimaryWriteRejected	PrimaryWriteRejected	Sum	ClientId, DomainName , NodeId
QueryCurrent	QueryCurrent	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
QueryFailure	QueryFailure	Sum	ClientId, DomainName , NodeId
QueryFailure-BeagleStone	QueryFailure-BeagleStone	Sum	ClientId, DomainName
QuerySuccess	QuerySuccess	Sum	ClientId, DomainName , NodeId
QuerySuccess-BeagleStone	QuerySuccess-BeagleStone	Sum	ClientId, DomainName
QueryTime InMillis	QueryTime InMillis	Histogram	ClientId, DomainName , NodeId
QueryTotal	QueryTotal	Sum	ClientId, DomainName , NodeId
ReadIOPS	ReadIOPS	Histogram	ClientId, DomainName , NodeId
ReadIOPSM icroBursting	ReadIOPSM icroBursting	Histogram	ClientId, DomainName , NodeId
ReadLatency	ReadLatency	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReadThroughput	ReadThroughput	Histogram	ClientId, DomainName , NodeId
ReadThroughputMicroBursting	ReadThroughputMicroBursting	Histogram	ClientId, DomainName , NodeId
RemoteStorageUsedSpace	RemoteStorageUsedSpace	Histogram	ClientId, DomainName
RemoteStorageWriteRejected	RemoteStorageWriteRejected	Sum	ClientId, DomainName , NodeId
ReplicaWriteRejected	ReplicaWriteRejected	Sum	ClientId, DomainName , NodeId
ReplicationLagMaxTime	ReplicationLagMaxTime	Histogram	ClientId, DomainName , NodeId
ReplicationNumBootstrappingIndices	ReplicationNumBootstrappingIndices	Histogram	ClientId, DomainName
ReplicationNumFailedIndices	ReplicationNumFailedIndices	Histogram	ClientId, DomainName
ReplicationNumIndexTasks	ReplicationNumIndexTasks	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReplicationNumPausedIndices	ReplicationNumPausedIndices	Histogram	ClientId, DomainName
ReplicationNumShardTasks	ReplicationNumShardTasks	Histogram	ClientId, DomainName
ReplicationNumSyncingIndices	ReplicationNumSyncingIndices	Histogram	ClientId, DomainName
ReplicationRate	ReplicationRate	Histogram	ClientId, DomainName , LeaderAlias
SQLDefaultCursorRequestCount	SQLDefaultCursorRequestCount	Sum	ClientId, DomainName
SQLFailedRequestCountByCusErr	SQLFailedRequestCountByCusErr	Sum	ClientId, DomainName
SQLFailedRequestCountBySysErr	SQLFailedRequestCountBySysErr	Sum	ClientId, DomainName
SQLRequestCount	SQLRequestCount	Sum	ClientId, DomainName
SQLUnhealthy	SQLUnhealthy	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ScrollCurrent	ScrollCurrent	Histogram	ClientId, DomainName , NodeId
ScrollCurrent-BeagleStone	ScrollCurrent-BeagleStone	Histogram	ClientId, DomainName
ScrollTotal	ScrollTotal	Sum	ClientId, DomainName , NodeId
ScrollTotal-BeagleStone	ScrollTotal-BeagleStone	Sum	ClientId, DomainName
SearchIdleReactivateCountTotal	SearchIdleReactivateCountTotal	Sum	ClientId, DomainName , NodeId
SearchLatency	SearchLatency	Histogram	ClientId, DomainName , NodeId
SearchPipelineRequestFailure	SearchPipelineRequestFailure	Sum	ClientId, DomainName , Processor
SearchPipelineRequestTime	SearchPipelineRequestTime	Histogram	ClientId, DomainName
SearchPipelineUnhealthy	SearchPipelineUnhealthy	Histogram	ClientId, DomainName
SearchRate	SearchRate	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SearchShardTaskCancelled	SearchShardTaskCancelled	Sum	ClientId, DomainName , NodeId
SearchTaskCancelled	SearchTaskCancelled	Sum	ClientId, DomainName , NodeId
SearchableDocuments	SearchableDocuments	Histogram	ClientId, DomainName
SegmentCount	SegmentCount	Histogram	ClientId, DomainName , NodeId
ShardCount	ShardCount	Histogram	ClientId, DomainName , NodeId, ShardRole
Shards.active	Shards.active	Histogram	ClientId, DomainName
Shards.activePrimary	Shards.activePrimary	Histogram	ClientId, DomainName
Shards.delayedUnassigned	Shards.delayedUnassigned	Histogram	ClientId, DomainName
Shards.initializing	Shards.initializing	Histogram	ClientId, DomainName
Shards.relocating	Shards.relocating	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Shards.unassigned	Shards.unassigned	Histogram	ClientId, DomainName
SingleStreamForecasterCount	SingleStreamForecasterCount	Histogram	ClientId, DomainName
SnapshotCount	SnapshotCount	Histogram	ClientId, DomainName , Repository
SnapshotFailures	SnapshotFailures	Sum	ClientId, DomainName , Repository
SoftWorkloadGroupCount	SoftWorkloadGroupCount	Histogram	ClientId, DomainName
SysMemoryUtilization	SysMemoryUtilization	Histogram	ClientId, DomainName , NodeId
TLSNegotiationError	TLSNegotiationError	Sum	ClientId, DomainName
ThreadCount	ThreadCount	Histogram	ClientId, DomainName
ThreadpoolBulkQueue	ThreadpoolBulkQueue	Histogram	ClientId, DomainName , NodeId
ThreadpoolBulkRejected	ThreadpoolBulkRejected	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Threadpool BulkThreads	Threadpool BulkThreads	Histogram	ClientId, DomainName , NodeId
Threadpool Force_merge Queue	Threadpool Force_merge Queue	Histogram	ClientId, DomainName , NodeId
Threadpool Force_merge Rejected	Threadpool Force_merge Rejected	Sum	ClientId, DomainName , NodeId
Threadpool Force_merge Threads	Threadpool Force_merge Threads	Histogram	ClientId, DomainName , NodeId
Threadpool IndexQueue	Threadpool IndexQueue	Histogram	ClientId, DomainName , NodeId
Threadpool IndexRejected	Threadpool IndexRejected	Sum	ClientId, DomainName , NodeId
Threadpool IndexSearcher Queue	Threadpool IndexSearcher Queue	Histogram	ClientId, DomainName , NodeId
Threadpool IndexSearcher Rejected	Threadpool IndexSearcher Rejected	Sum	ClientId, DomainName , NodeId
Threadpool IndexSearcher Threads	Threadpool IndexSearcher Threads	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Threadpool IndexThreads	Threadpool IndexThreads	Histogram	ClientId, DomainName , NodeId
Threadpool MergeQueue	Threadpool MergeQueue	Histogram	ClientId, DomainName , NodeId
Threadpool MergeRejected	Threadpool MergeRejected	Sum	ClientId, DomainName , NodeId
Threadpool MergeThreads	Threadpool MergeThreads	Histogram	ClientId, DomainName , NodeId
Threadpool Opendistro_monitor_runnerQueue	Threadpool Opendistro_monitor_runnerQueue	Histogram	ClientId, DomainName , NodeId
Threadpool Opendistro_monitor_runnerRejected	Threadpool Opendistro_monitor_runnerRejected	Sum	ClientId, DomainName , NodeId
Threadpool Opendistro_monitor_runnerThreads	Threadpool Opendistro_monitor_runnerThreads	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Threadpool SearchQueue	Threadpool SearchQueue	Histogram	ClientId, DomainName , NodeId
Threadpool SearchRejected	Threadpool SearchRejected	Sum	ClientId, DomainName , NodeId
Threadpool SearchThreads	Threadpool SearchThreads	Histogram	ClientId, DomainName , NodeId
Threadpool WriteQueue	Threadpool WriteQueue	Histogram	ClientId, DomainName , NodeId
Threadpool WriteRejected	Threadpool WriteRejected	Sum	ClientId, DomainName , NodeId
Threadpool WriteThreads	Threadpool WriteThreads	Histogram	ClientId, DomainName , NodeId
Threadpoolsql-workerQueue	Threadpoolsql-workerQueue	Histogram	ClientId, DomainName , NodeId
Threadpoolsql-workerRejected	Threadpoolsql-workerRejected	Sum	ClientId, DomainName , NodeId
Threadpoolsql-workerThreads	Threadpoolsql-workerThreads	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ThroughputThrottle	ThroughputThrottle	Histogram	ClientId, DomainName , NodeId
TookCurrent	TookCurrent	Histogram	ClientId, DomainName , NodeId
TookTimeInMillis	TookTimeInMillis	Histogram	ClientId, DomainName , NodeId
TookTotal	TookTotal	Sum	ClientId, DomainName , NodeId
TotalPointInTime	TotalPointInTime	Histogram	ClientId, DomainName , NodeId
TotalRemoteReindexCallsFailed	TotalRemoteReindexCallsFailed	Sum	ClientId, DomainName , RemoteDomain
TotalRemoteReindexCallsSucceeded	TotalRemoteReindexCallsSucceeded	Sum	ClientId, DomainName , RemoteDomain
VolumeStalledIOCheck	VolumeStalledIOCheck	Histogram	ClientId, DomainName , NodeId
WarmCPUUtilization	WarmCPUUtilization	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WarmConcurrentSearchLatency	WarmConcurrentSearchLatency	Histogram	ClientId, DomainName , NodeId
WarmConcurrentSearchRate	WarmConcurrentSearchRate	Sum	ClientId, DomainName , NodeId
WarmFreeStorageSpace	WarmFreeStorageSpace	Histogram	ClientId, DomainName , NodeId
WarmJVMGCOldCollectionCount	WarmJVMGCOldCollectionCount	Sum	ClientId, DomainName , NodeId
WarmJVMGCOldCollectionTime	WarmJVMGCOldCollectionTime	Sum	ClientId, DomainName , NodeId
WarmJVMGCYoungCollectionCount	WarmJVMGCYoungCollectionCount	Sum	ClientId, DomainName , NodeId
WarmJVMGCYoungCollectionTime	WarmJVMGCYoungCollectionTime	Sum	ClientId, DomainName , NodeId
WarmJVMMemoryPressure	WarmJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
WarmNodes	WarmNodes	Histogram	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WarmOldGenJVMMemoryPressure	WarmOldGenJVMMemoryPressure	Histogram	ClientId, DomainName , NodeId
WarmSearchLatency	WarmSearchLatency	Histogram	ClientId, DomainName , NodeId
WarmSearchRate	WarmSearchRate	Sum	ClientId, DomainName , NodeId
WarmSearchableDocuments	WarmSearchableDocuments	Histogram	ClientId, DomainName
WarmStorageSpaceUtilization	WarmStorageSpaceUtilization	Histogram	ClientId, DomainName , NodeId
WarmSysMemoryUtilization	WarmSysMemoryUtilization	Histogram	ClientId, DomainName , NodeId
WarmThreadpoolIndexSearcherQueue	WarmThreadpoolIndexSearcherQueue	Histogram	ClientId, DomainName , NodeId
WarmThreadpoolIndexSearcherRejected	WarmThreadpoolIndexSearcherRejected	Sum	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WarmThreadpoolIndexSearcherThreads	WarmThreadpoolIndexSearcherThreads	Histogram	ClientId, DomainName , NodeId
WarmThreadpoolSearchQueue	WarmThreadpoolSearchQueue	Histogram	ClientId, DomainName , NodeId
WarmThreadpoolSearchRejected	WarmThreadpoolSearchRejected	Sum	ClientId, DomainName , NodeId
WarmThreadpoolSearchThreads	WarmThreadpoolSearchThreads	Histogram	ClientId, DomainName , NodeId
WarmToColdMigrationFailureCount	WarmToColdMigrationFailureCount	Sum	ClientId, DomainName
WarmToColdMigrationLatency	WarmToColdMigrationLatency	Histogram	ClientId, DomainName
WarmToColdMigrationQueueSize	WarmToColdMigrationQueueSize	Histogram	ClientId, DomainName
WarmToColdMigrationSuccessCount	WarmToColdMigrationSuccessCount	Sum	ClientId, DomainName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WarmToHotMigrationQueueSize	WarmToHotMigrationQueueSize	Histogram	ClientId, DomainName
WorkloadCPUCancellations	WorkloadCPUCancellations	Sum	ClientId, DomainName
WorkloadCPURejections	WorkloadCPURejections	Sum	ClientId, DomainName
WorkloadManagementEnabled	WorkloadManagementEnabled	Histogram	ClientId, DomainName
WorkloadMemoryCancellations	WorkloadMemoryCancellations	Sum	ClientId, DomainName
WorkloadQueryCompletions	WorkloadQueryCompletions	Sum	ClientId, DomainName
WriteIOPS	WriteIOPS	Histogram	ClientId, DomainName , NodeId
WriteIOPSMicroBursting	WriteIOPSMicroBursting	Histogram	ClientId, DomainName , NodeId
WriteLatency	WriteLatency	Histogram	ClientId, DomainName , NodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WriteThroughput	WriteThroughput	Histogram	ClientId, DomainName , NodeId
WriteThroughputMicroBursting	WriteThroughputMicroBursting	Histogram	ClientId, DomainName , NodeId
warmQueryFailure	warmQueryFailure	Sum	ClientId, DomainName , NodeId
warmQuerySuccess	warmQuerySuccess	Sum	ClientId, DomainName , NodeId

AWS/ElastiCache

The AWS/ElastiCache namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticache. All enriched instruments use Delta temporality.

- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::ReplicationGroup
- AWS::ElastiCache::ServerlessCache

AWS::ElastiCache::CacheCluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveDefragHits	ActiveDefragHits	Sum	CacheClusterId , CacheNodeId
AllocatorFragmentationBytes	AllocatorFragmentationBytes	Histogram	CacheClusterId , CacheNodeId
AllocatorFragmentationRatio	AllocatorFragmentationRatio	Histogram	CacheClusterId , CacheNodeId
AuthenticationFailures	AuthenticationFailures	Sum	CacheClusterId , CacheNodeId
BlockedConnections	BlockedConnections	Histogram	CacheClusterId , CacheNodeId
BytesReadFromDisk	BytesReadFromDisk	Sum	CacheClusterId , CacheNodeId
BytesReadIntoMemcached	BytesReadIntoMemcached	Sum	CacheClusterId , CacheNodeId
BytesUsedForCache	BytesUsedForCache	Histogram	CacheClusterId , CacheNodeId
BytesUsedForCacheItems	BytesUsedForCacheItems	Histogram	CacheClusterId , CacheNodeId
BytesUsedForHash	BytesUsedForHash	Histogram	CacheClusterId , CacheNodeId
BytesWrittenOutFromMemcached	BytesWrittenOutFromMemcached	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesWrittenToDisk	BytesWrittenToDisk	Sum	CacheClusterId , CacheNodeId
CPUCreditBalance	CPUCreditBalance	Histogram	CacheClusterId , CacheNodeId
CPUCreditUsage	CPUCreditUsage	Sum	CacheClusterId , CacheNodeId
CPUUtilization	CPUUtilization	Histogram	CacheClusterId , CacheNodeId
CacheHitRate	CacheHitRate	Histogram	CacheClusterId , CacheNodeId
CacheHits	CacheHits	Sum	CacheClusterId , CacheNodeId
CacheMisses	CacheMisses	Sum	CacheClusterId , CacheNodeId
CasBadval	CasBadval	Sum	CacheClusterId , CacheNodeId
CasHits	CasHits	Sum	CacheClusterId , CacheNodeId
CasMisses	CasMisses	Sum	CacheClusterId , CacheNodeId
ChannelAuthorizationFailures	ChannelAuthorizationFailures	Sum	CacheClusterId , CacheNodeId
ClusterBasedCmds	ClusterBasedCmds	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ClusterBasedCmdsLatency	ClusterBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
CmdConfigGet	CmdConfigGet	Sum	CacheClusterId , CacheNodeId
CmdConfigSet	CmdConfigSet	Sum	CacheClusterId , CacheNodeId
CmdFlush	CmdFlush	Sum	CacheClusterId , CacheNodeId
CmdGet	CmdGet	Sum	CacheClusterId , CacheNodeId
CmdSet	CmdSet	Sum	CacheClusterId , CacheNodeId
CmdTouch	CmdTouch	Sum	CacheClusterId , CacheNodeId
CommandAuthorizationFailures	CommandAuthorizationFailures	Sum	CacheClusterId , CacheNodeId
CurrConfig	CurrConfig	Histogram	CacheClusterId , CacheNodeId
CurrConnections	CurrConnections	Histogram	CacheClusterId , CacheNodeId
CurrItems	CurrItems	Histogram	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CurrItemsWithVolatileFields	CurrItemsWithVolatileFields	Histogram	CacheClusterId , CacheNodeId
CurrVolatileItems	CurrVolatileItems	Histogram	CacheClusterId , CacheNodeId
DB0AverageTTL	DB0AverageTTL	Histogram	CacheClusterId , CacheNodeId
DatabaseAuthorizationFailures	DatabaseAuthorizationFailures	Sum	CacheClusterId , CacheNodeId
DatabaseCapacityUsageCountedForEvictPercentage	DatabaseCapacityUsageCountedForEvictPercentage	Histogram	CacheClusterId , CacheNodeId
DatabaseCapacityUsagePercentage	DatabaseCapacityUsagePercentage	Histogram	CacheClusterId , CacheNodeId
DatabaseMemoryUsageCountedForEvictPercentage	DatabaseMemoryUsageCountedForEvictPercentage	Histogram	CacheClusterId , CacheNodeId
DatabaseMemoryUsagePercentage	DatabaseMemoryUsagePercentage	Histogram	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DecrHits	DecrHits	Sum	CacheClusterId , CacheNodeId
DecrMisses	DecrMisses	Sum	CacheClusterId , CacheNodeId
DeleteHits	DeleteHits	Sum	CacheClusterId , CacheNodeId
DeleteMisses	DeleteMisses	Sum	CacheClusterId , CacheNodeId
DurabilityBufferExceededErrorCount	DurabilityBufferExceededErrorCount	Sum	CacheClusterId , CacheNodeId
DurabilityLag	DurabilityLag	Histogram	CacheClusterId , CacheNodeId
EngineCPUUtilization	EngineCPUUtilization	Histogram	CacheClusterId , CacheNodeId
ErrorCount	ErrorCount	Sum	CacheClusterId , CacheNodeId
EvalBasedCmds	EvalBasedCmds	Sum	CacheClusterId , CacheNodeId
EvalBasedCmdsLatency	EvalBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
EvictedUnfetched	EvictedUnfetched	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Evictions	Evictions	Sum	CacheClusterId , CacheNodeId
ExpiredUnfetched	ExpiredUnfetched	Sum	CacheClusterId , CacheNodeId
FreeableMemory	FreeableMemory	Histogram	CacheClusterId , CacheNodeId
GeoSpatialBasedCmds	GeoSpatialBasedCmds	Sum	CacheClusterId , CacheNodeId
GeoSpatialBasedCmdsLatency	GeoSpatialBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
GetHits	GetHits	Sum	CacheClusterId , CacheNodeId
GetMisses	GetMisses	Sum	CacheClusterId , CacheNodeId
GetTypeCmds	GetTypeCmds	Sum	CacheClusterId , CacheNodeId
GetTypeCmdsLatency	GetTypeCmdsLatency	Histogram	CacheClusterId , CacheNodeId
HashBasedCmds	HashBasedCmds	Sum	CacheClusterId , CacheNodeId
HashBasedCmdsLatency	HashBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
HyperLogLogBasedCmds	HyperLogLogBasedCmds	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HyperLogLogBasedCmdsLatency	HyperLogLogBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
IamAuthenticationExpirations	IamAuthenticationExpirations	Sum	CacheClusterId , CacheNodeId
IamAuthenticationThrottling	IamAuthenticationThrottling	Sum	CacheClusterId , CacheNodeId
IncrHits	IncrHits	Sum	CacheClusterId , CacheNodeId
IncrMisses	IncrMisses	Sum	CacheClusterId , CacheNodeId
IsMaster	IsMaster	Histogram	CacheClusterId , CacheNodeId
JsonBasedCmds	JsonBasedCmds	Sum	CacheClusterId , CacheNodeId
JsonBasedCmdsLatency	JsonBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
JsonBasedGetCmds	JsonBasedGetCmds	Sum	CacheClusterId , CacheNodeId
JsonBasedGetCmdsLatency	JsonBasedGetCmdsLatency	Histogram	CacheClusterId , CacheNodeId
JsonBasedSetCmds	JsonBasedSetCmds	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
JsonBasedSetCmdsLatency	JsonBasedSetCmdsLatency	Histogram	CacheClusterId , CacheNodeId
KeyAuthorizationFailures	KeyAuthorizationFailures	Sum	CacheClusterId , CacheNodeId
KeyBasedCmds	KeyBasedCmds	Sum	CacheClusterId , CacheNodeId
KeyBasedCmdsLatency	KeyBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
KeysTracked	KeysTracked	Histogram	CacheClusterId , CacheNodeId
ListBasedCmds	ListBasedCmds	Sum	CacheClusterId , CacheNodeId
ListBasedCmdsLatency	ListBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
MajorPageFaults	MajorPageFaults	Sum	CacheClusterId , CacheNodeId
MasterLinkHealthStatus	MasterLinkHealthStatus	Histogram	CacheClusterId , CacheNodeId
MemoryFragmentationRatio	MemoryFragmentationRatio	Histogram	CacheClusterId , CacheNodeId
NetworkBandwidthInAllowanceExceeded	NetworkBandwidthInAllowanceExceeded	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkBandwidthOutAllowanceExceeded	NetworkBandwidthOutAllowanceExceeded	Sum	CacheClusterId , CacheNodeId
NetworkBaselineMaxUsageInPercentage	NetworkBaselineMaxUsageInPercentage	Histogram	CacheClusterId , CacheNodeId
NetworkBaselineMaxUsageOutPercentage	NetworkBaselineMaxUsageOutPercentage	Histogram	CacheClusterId , CacheNodeId
NetworkBaselineUsageInPercentage	NetworkBaselineUsageInPercentage	Histogram	CacheClusterId , CacheNodeId
NetworkBaselineUsageOutPercentage	NetworkBaselineUsageOutPercentage	Histogram	CacheClusterId , CacheNodeId
NetworkBytesIn	NetworkBytesIn	Sum	CacheClusterId , CacheNodeId
NetworkBytesOut	NetworkBytesOut	Sum	CacheClusterId , CacheNodeId
NetworkConntrackAllowanceExceeded	NetworkConntrackAllowanceExceeded	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkLinkLocalAllowanceExceeded	NetworkLinkLocalAllowanceExceeded	Sum	CacheClusterId , CacheNodeId
NetworkMaxBytesIn	NetworkMaxBytesIn	Histogram	CacheClusterId , CacheNodeId
NetworkMaxBytesOut	NetworkMaxBytesOut	Histogram	CacheClusterId , CacheNodeId
NetworkMaxPacketsIn	NetworkMaxPacketsIn	Histogram	CacheClusterId , CacheNodeId
NetworkMaxPacketsOut	NetworkMaxPacketsOut	Histogram	CacheClusterId , CacheNodeId
NetworkPacketsIn	NetworkPacketsIn	Sum	CacheClusterId , CacheNodeId
NetworkPacketsOut	NetworkPacketsOut	Sum	CacheClusterId , CacheNodeId
NetworkPacketsPerSecondAllowanceExceeded	NetworkPacketsPerSecondAllowanceExceeded	Sum	CacheClusterId , CacheNodeId
NewConnections	NewConnections	Sum	CacheClusterId , CacheNodeId
NewItems	NewItems	Sum	CacheClusterId , CacheNodeId
NonKeyTypeCmds	NonKeyTypeCmds	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NonKeyTypeCmdsLatency	NonKeyTypeCmdsLatency	Histogram	CacheClusterId , CacheNodeId
NumItemsReadFromDisk	NumItemsReadFromDisk	Sum	CacheClusterId , CacheNodeId
NumItemsWrittenToDisk	NumItemsWrittenToDisk	Sum	CacheClusterId , CacheNodeId
ProcessedCommands	ProcessedCommands	Sum	CacheClusterId , CacheNodeId
PubSubBasedCmds	PubSubBasedCmds	Sum	CacheClusterId , CacheNodeId
PubSubBasedCmdsLatency	PubSubBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
PubSubChannels	PubSubChannels	Histogram	CacheClusterId , CacheNodeId
PubSubShardChannels	PubSubShardChannels	Histogram	CacheClusterId , CacheNodeId
Reclaimed	Reclaimed	Sum	CacheClusterId , CacheNodeId
ReclaimedFields	ReclaimedFields	Sum	CacheClusterId , CacheNodeId
RejectedConnections	RejectedConnections	Histogram	CacheClusterId , CacheNodeId
ReplicationBytes	ReplicationBytes	Sum	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReplicationLag	ReplicationLag	Histogram	CacheClusterId , CacheNodeId
SaveInProgress	SaveInProgress	Histogram	CacheClusterId , CacheNodeId
SearchBasedCmds	SearchBasedCmds	Sum	CacheClusterId , CacheNodeId
SearchBasedCmdsLatency	SearchBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
SearchBasedGetCmds	SearchBasedGetCmds	Sum	CacheClusterId , CacheNodeId
SearchBasedGetCmdsLatency	SearchBasedGetCmdsLatency	Histogram	CacheClusterId , CacheNodeId
SearchBasedSetCmds	SearchBasedSetCmds	Sum	CacheClusterId , CacheNodeId
SearchBasedSetCmdsLatency	SearchBasedSetCmdsLatency	Histogram	CacheClusterId , CacheNodeId
SearchNumberOfIndexes	SearchNumberOfIndexes	Histogram	CacheClusterId , CacheNodeId
SearchTotalIndexedDocuments	SearchTotalIndexedDocuments	Histogram	CacheClusterId , CacheNodeId
SearchUsedMemoryBytes	SearchUsedMemoryBytes	Histogram	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SetBasedCmds	SetBasedCmds	Sum	CacheClusterId , CacheNodeId
SetBasedCmdsLatency	SetBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
SetTypeCmds	SetTypeCmds	Sum	CacheClusterId , CacheNodeId
SetTypeCmdsLatency	SetTypeCmdsLatency	Histogram	CacheClusterId , CacheNodeId
SlabsMoved	SlabsMoved	Sum	CacheClusterId , CacheNodeId
SortedSetBasedCmds	SortedSetBasedCmds	Sum	CacheClusterId , CacheNodeId
SortedSetBasedCmdsLatency	SortedSetBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
StreamBasedCmds	StreamBasedCmds	Sum	CacheClusterId , CacheNodeId
StreamBasedCmdsLatency	StreamBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId
StringBasedCmds	StringBasedCmds	Sum	CacheClusterId , CacheNodeId
StringBasedCmdsLatency	StringBasedCmdsLatency	Histogram	CacheClusterId , CacheNodeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SuccessfulReadRequestLatency	SuccessfulReadRequestLatency	Histogram	CacheClusterId , CacheNodeId
SuccessfulWriteRequestLatency	SuccessfulWriteRequestLatency	Histogram	CacheClusterId , CacheNodeId
SwapUsage	SwapUsage	Histogram	CacheClusterId , CacheNodeId
TouchHits	TouchHits	Sum	CacheClusterId , CacheNodeId
TouchMisses	TouchMisses	Sum	CacheClusterId , CacheNodeId
TrafficManagementActive	TrafficManagementActive	Histogram	CacheClusterId , CacheNodeId
UnusedMemory	UnusedMemory	Histogram	CacheClusterId , CacheNodeId
UsedMemoryDataset	UsedMemoryDataset	Histogram	CacheClusterId , CacheNodeId

AWS::ElastiCache::ReplicationGroup

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DatabaseCapacityUsageCountedForEvictPercentage	DatabaseCapacityUsageCountedForEvictPercentage	Histogram	NodeGroupId ,ReplicationGroupId
DatabaseMemoryUsageCountedForEvictPercentage	DatabaseMemoryUsageCountedForEvictPercentage	Histogram	NodeGroupId ,ReplicationGroupId

AWS::ElastiCache::ServerlessCache

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ClusterBasedCmdsECPUs	ClusterBasedCmdsECPUs	Sum	clusterId
ElastiCacheProcessingUnits	ElastiCacheProcessingUnits	Sum	clusterId
EvalBasedCmdsECPUs	EvalBasedCmdsECPUs	Sum	clusterId
GeoSpatialBasedCmdsECPUs	GeoSpatialBasedCmdsECPUs	Sum	clusterId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GetTypeCmdsECPUs	GetTypeCmdsECPUs	Sum	clusterId
HashBasedCmdsECPUs	HashBasedCmdsECPUs	Sum	clusterId
HyperLogLogBasedCmdsECPUs	HyperLogLogBasedCmdsECPUs	Sum	clusterId
JsonBasedCmdsECPUs	JsonBasedCmdsECPUs	Sum	clusterId
KeyBasedCmdsECPUs	KeyBasedCmdsECPUs	Sum	clusterId
ListBasedCmdsECPUs	ListBasedCmdsECPUs	Sum	clusterId
NonKeyTypeCmdsECPUs	NonKeyTypeCmdsECPUs	Sum	clusterId
PubSubBasedCmdsECPUs	PubSubBasedCmdsECPUs	Sum	clusterId
SetBasedCmdsECPUs	SetBasedCmdsECPUs	Sum	clusterId
SetTypeCmdsECPUs	SetTypeCmdsECPUs	Sum	clusterId
SortedSetBasedCmdsECPUs	SortedSetBasedCmdsECPUs	Sum	clusterId
StreamBasedCmdsECPUs	StreamBasedCmdsECPUs	Sum	clusterId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StringBasedCmdsECPUs	StringBasedCmdsECPUs	Sum	clusterId
ThrottledCmds	ThrottledCmds	Sum	clusterId
TotalCmdsCount	TotalCmdsCount	Sum	clusterId

AWS/ElasticBeanstalk

The AWS/ElasticBeanstalk namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticbeanstalk. All enriched instruments use Delta temporality.

- AWS::ElasticBeanstalk::Environment

AWS::ElasticBeanstalk::Environment

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ApplicationLatencyP10	ApplicationLatencyP10	Histogram	EnvironmentName, InstanceId
ApplicationLatencyP50	ApplicationLatencyP50	Histogram	EnvironmentName, InstanceId
ApplicationLatencyP75	ApplicationLatencyP75	Histogram	EnvironmentName, InstanceId
ApplicationLatencyP85	ApplicationLatencyP85	Histogram	EnvironmentName, InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Applicati onLatencyP90	Applicati onLatencyP90	Histogram	EnvironmentName , InstanceId
Applicati onLatencyP95	Applicati onLatencyP95	Histogram	EnvironmentName , InstanceId
Applicati onLatencyP99	Applicati onLatencyP99	Histogram	EnvironmentName , InstanceId
Applicati onLatencyP99.9	Applicati onLatencyP99.9	Histogram	EnvironmentName , InstanceId
Applicati onRequests2xx	Applicati onRequests2xx	Sum	EnvironmentName , InstanceId
Applicati onRequests3xx	Applicati onRequests3xx	Sum	EnvironmentName , InstanceId
Applicati onRequests4xx	Applicati onRequests4xx	Sum	EnvironmentName , InstanceId
Applicati onRequests5xx	Applicati onRequests5xx	Sum	EnvironmentName , InstanceId
Applicati onRequest sTotal	Applicati onRequest sTotal	Sum	EnvironmentName , InstanceId
CPUIdle	CPUIdle	Histogram	EnvironmentName , InstanceId
CPUIowait	CPUIowait	Histogram	EnvironmentName , InstanceId
CPUIrq	CPUIrq	Histogram	EnvironmentName , InstanceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUNice	CPUNice	Histogram	EnvironmentName , InstanceId
CPUPrivileged	CPUPrivileged	Histogram	EnvironmentName , InstanceId
CPUSoftirq	CPUSoftirq	Histogram	EnvironmentName , InstanceId
CPUSystem	CPUSystem	Histogram	EnvironmentName , InstanceId
CPUUser	CPUUser	Histogram	EnvironmentName , InstanceId
InstanceHealth	InstanceHealth	Histogram	EnvironmentName , InstanceId
LoadAverage1min	LoadAverage1min	Histogram	EnvironmentName , InstanceId
LoadAverage5min	LoadAverage5min	Histogram	EnvironmentName , InstanceId
RootFilesystemUtil	RootFilesystemUtil	Histogram	EnvironmentName , InstanceId

AWS/ElasticMapReduce

The AWS/ElasticMapReduce namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticmapreduce. All enriched instruments use Delta temporality.

- AWS::EMR::Cluster

AWS::EMR::Cluster

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AppsCompleted	AppsCompleted	Sum	JobFlowId
AppsFailed	AppsFailed	Sum	JobFlowId
AppsKilled	AppsKilled	Sum	JobFlowId
AppsPending	AppsPending	Histogram	JobFlowId
AppsRunning	AppsRunning	Histogram	JobFlowId
AppsSubmitted	AppsSubmitted	Sum	JobFlowId
AutoTerminationIsClusterIdle	AutoTerminationIsClusterIdle	Histogram	JobFlowId
ContainerAllocated	ContainerAllocated	Histogram	JobFlowId
ContainerPending	ContainerPending	Histogram	JobFlowId
ContainerPendingRatio	ContainerPendingRatio	Histogram	JobFlowId
ContainerReserved	ContainerReserved	Histogram	JobFlowId
CoreNodesPending	CoreNodesPending	Histogram	JobFlowId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CoreNodes Requested	CoreNodes Requested	Histogram	JobFlowId
CoreNodes Running	CoreNodes Running	Histogram	JobFlowId
CoreUnits Requested	CoreUnits Requested	Histogram	JobFlowId
CoreUnits Running	CoreUnits Running	Histogram	JobFlowId
CoreVCPUR equested	CoreVCPUR equested	Histogram	JobFlowId
CoreVCPURunning	CoreVCPURunning	Histogram	JobFlowId
HDFSBytesRead	HDFSBytesRead	Sum	JobFlowId
HDFSBytes Written	HDFSBytes Written	Sum	JobFlowId
HDFSUtilization	HDFSUtilization	Histogram	JobFlowId
IsIdle	IsIdle	Histogram	JobFlowId
JobsFailed	JobsFailed	Sum	JobFlowId
JobsRunning	JobsRunning	Histogram	JobFlowId
LiveDataNodes	LiveDataNodes	Histogram	JobFlowId
LiveTaskT rackers	LiveTaskT rackers	Histogram	JobFlowId
MRActiveNodes	MRActiveNodes	Histogram	JobFlowId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MRDecommissionedNodes	MRDecommissionedNodes	Histogram	JobFlowId
MRLostNodes	MRLostNodes	Histogram	JobFlowId
MRTotalNodes	MRTotalNodes	Histogram	JobFlowId
MRUnhealthyNodes	MRUnhealthyNodes	Histogram	JobFlowId
MapSlotsOpen	MapSlotsOpen	Histogram	JobFlowId
MapTasksRemaining	MapTasksRemaining	Histogram	JobFlowId
MapTasksRunning	MapTasksRunning	Histogram	JobFlowId
MissingBlocks	MissingBlocks	Histogram	JobFlowId
ReduceSlotsOpen	ReduceSlotsOpen	Histogram	JobFlowId
ReduceTasksRemaining	ReduceTasksRemaining	Histogram	JobFlowId
ReduceTasksRunning	ReduceTasksRunning	Histogram	JobFlowId
RemainingMapTasksPerSlot	RemainingMapTasksPerSlot	Histogram	JobFlowId
S3BytesRead	S3BytesRead	Sum	JobFlowId
S3BytesWritten	S3BytesWritten	Sum	JobFlowId
TaskNodesPending	TaskNodesPending	Histogram	JobFlowId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TaskNodes Requested	TaskNodes Requested	Histogram	JobFlowId
TaskNodes Running	TaskNodes Running	Histogram	JobFlowId
TaskUnits Requested	TaskUnits Requested	Histogram	JobFlowId
TaskUnits Running	TaskUnits Running	Histogram	JobFlowId
TaskVCPUR equested	TaskVCPUR equested	Histogram	JobFlowId
TaskVCPURunning	TaskVCPURunning	Histogram	JobFlowId
TotalLoad	TotalLoad	Histogram	JobFlowId
TotalNode sRequested	TotalNode sRequested	Histogram	JobFlowId
TotalNode sRunning	TotalNode sRunning	Histogram	JobFlowId
TotalNote bookKernels	TotalNote bookKernels	Histogram	JobFlowId
TotalUnit sRequested	TotalUnit sRequested	Histogram	JobFlowId
TotalUnit sRunning	TotalUnit sRunning	Histogram	JobFlowId
TotalVCPU Requested	TotalVCPU Requested	Histogram	JobFlowId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalVCPU Running	TotalVCPU Running	Histogram	JobFlowId

AWS/EventBridge/Pipes

The AWS/EventBridge/Pipes namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/pipes. All enriched instruments use Delta temporality.

- AWS::Pipes::Pipe

AWS::Pipes::Pipe

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Duration	Duration	Histogram	PipeName
Enrichmen tStageDuration	Enrichmen tStageDuration	Histogram	PipeName
Enrichmen tStageFailed	Enrichmen tStageFailed	Sum	PipeName
EventCount	EventCount	Sum	PipeName
EventSize	EventSize	Histogram	PipeName
ExecutionFailed	ExecutionFailed	Sum	PipeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ExecutionPartiallyFailed	ExecutionPartiallyFailed	Sum	PipeName
ExecutionTimeout	ExecutionTimeout	Sum	PipeName
TargetStageDuration	TargetStageDuration	Histogram	PipeName
TargetStageFailed	TargetStageFailed	Sum	PipeName
TargetStagePartiallyFailed	TargetStagePartiallyFailed	Sum	PipeName
TargetStageSkipped	TargetStageSkipped	Sum	PipeName

AWS/Events

The `AWS/Events` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/events`. All enriched instruments use Delta temporality.

- AWS::Events::Rule
- `event-source`

AWS::Events::Rule

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FailedInvocations	FailedInvocations	Sum	EventBusName , RuleName
IngestiontoInvocationCompleteLatency	IngestiontoInvocationCompleteLatency	Histogram	EventBusName , RuleName
IngestiontoInvocationStartLatency	IngestiontoInvocationStartLatency	Histogram	EventBusName , RuleName
Invocations	Invocations	Sum	EventBusName , RuleName
InvocationsSentToDlq	InvocationsSentToDlq	Sum	EventBusName , RuleName
ThrottledRules	ThrottledRules	Sum	EventBusName , RuleName
TriggeredRules	TriggeredRules	Sum	EventBusName , RuleName

`event-source`

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Events	Events	Sum	EventSourceName

AWS/FSx

The AWS/FSx namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/fsx. All enriched instruments use Delta temporality.

- AWS::FSx::FileSystem

AWS::FSx::FileSystem

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AgeOfOldestQueuedMessage	AgeOfOldestQueuedMessage	Histogram	FileSystemId , Publisher
CPUUtilization	CPUUtilization	Histogram	FileSystemId
CapacityPoolReadBytes	CapacityPoolReadBytes	Sum	FileSystemId , VolumeId
CapacityPoolReadOperations	CapacityPoolReadOperations	Sum	FileSystemId , VolumeId
CapacityPoolWriteBytes	CapacityPoolWriteBytes	Sum	FileSystemId , VolumeId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CapacityPoolWriteOperations	CapacityPoolWriteOperations	Sum	FileSystemId , VolumeId
ClientConnections	ClientConnections	Histogram	FileSystemId
CompressionRatio	CompressionRatio	Histogram	FileSystemId , VolumeId
DataReadBytes	DataReadBytes	Sum	FileSystemId , VolumeId
DataReadOperationTime	DataReadOperationTime	Sum	FileSystemId , VolumeId
DataReadOperations	DataReadOperations	Sum	FileSystemId , VolumeId
DataWriteBytes	DataWriteBytes	Sum	FileSystemId , VolumeId
DataWriteLatency	DataWriteLatency	Histogram	FileSystemId , StorageTargetId
DataWriteOperationTime	DataWriteOperationTime	Sum	FileSystemId , VolumeId
DataWriteOperations	DataWriteOperations	Sum	FileSystemId , VolumeId
DirectoryCreateOperations	DirectoryCreateOperations	Sum	FileSystemId , StorageTargetId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DirectoryCreateOperations	DirectoryCreateOperationsByJob	Sum	FileSystemId, JobId, StorageTargetId
DirectoryDeleteOperations	DirectoryDeleteOperations	Sum	FileSystemId, StorageTargetId
DirectoryDeleteOperations	DirectoryDeleteOperationsByJob	Sum	FileSystemId, JobId, StorageTargetId
DiskIopsExceededCheck	DiskIopsExceededCheck	Histogram	FileSystemId, StorageTargetId
DiskThroughputBalance	DiskThroughputBalance	Histogram	FileSystemId
DiskThroughputExceededCheck	DiskThroughputExceededCheck	Histogram	FileSystemId, StorageTargetId
DiskThroughputUtilization	DiskThroughputUtilization	Histogram	FileSystemId
FileCreateOperations	FileCreateOperations	Sum	FileSystemId, StorageTargetId
FileCreateOperations	FileCreateOperationsByJob	Sum	FileSystemId, JobId, StorageTargetId
FileDeleteOperations	FileDeleteOperations	Sum	FileSystemId, StorageTargetId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FileDeleteOperations	FileDeleteOperationsByJob	Sum	FileSystemId , JobId, StorageTargetId
FileOpenOperations	FileOpenOperations	Sum	FileSystemId , StorageTargetId
FileOpenOperations	FileOpenOperationsByJob	Sum	FileSystemId , JobId, StorageTargetId
FileServerDiskIopsBalance	FileServerDiskIopsBalance	Histogram	FileSystemId
FileServerDiskIopsUtilization	FileServerDiskIopsUtilization	Histogram	FileSystemId
FileServerDiskThroughputBalance	FileServerDiskThroughputBalance	Histogram	FileServer , FileSystemId
FileServerDiskThroughputUtilization	FileServerDiskThroughputUtilization	Histogram	FileSystemId
FilesCapacity	FilesCapacity	Histogram	FileSystemId , VolumeId
FilesUsed	FilesUsed	Histogram	FileSystemId , VolumeId
FreeDataStorageCapacity	FreeDataStorageCapacity	Histogram	FileSystemId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FreeStorageCapacity	FreeStorageCapacity	Histogram	FileSystemId
MemoryUtilization	MemoryUtilization	Histogram	FileSystemId
MetadataOperationTime	MetadataOperationTime	Sum	FileSystemId , VolumeId
MetadataOperations	MetadataOperations	Sum	FileSystemId , VolumeId
NetworkReceivedBytes	NetworkReceivedBytes	Sum	FileServer , FileSystemId
NetworkSentBytes	NetworkSentBytes	Sum	FileServer , FileSystemId
NetworkThroughputUtilization	NetworkThroughputUtilization	Histogram	FileSystemId
RenameOperations	RenameOperationsByJob	Sum	FileSystemId , JobId, StorageTargetId
RepositoryRenameOperations	RepositoryRenameOperations	Sum	FileSystemId , Publisher
StatOperations	StatOperationsByJob	Sum	FileSystemId , JobId, StorageTargetId

AWS/Firehose

The `AWS/Firehose` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/firehose`. All enriched instruments use Delta temporality.

- AWS::KinesisFirehose::DeliveryStream

AWS::KinesisFirehose::DeliveryStream

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BackupToS3.Bytes	BackupToS3.Bytes	Sum	DeliveryStreamName
BackupToS3.DataFreshness	BackupToS3.DataFreshness	Histogram	DeliveryStreamName
BackupToS3.Records	BackupToS3.Records	Sum	DeliveryStreamName
BackupToS3.Success	BackupToS3.Success	Sum	DeliveryStreamName
BytesPerSecondLimit	BytesPerSecondLimit	Histogram	DeliveryStreamName
DataReadFromKinesisStream.Bytes	DataReadFromKinesisStream.Bytes	Sum	DeliveryStreamName
DataReadFromKinesi	DataReadFromKinesi	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
sStream.Records	sStream.Records		
DataReadFromSource.Backpressured	DataReadFromSource.Backpressured	Sum	DeliveryStreamName
DataReadFromSource.Bytes	DataReadFromSource.Bytes	Sum	DeliveryStreamName
DataReadFromSource.Records	DataReadFromSource.Records	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.AuthFailure	DeliveryToAmazonOpenSearchServerless.AuthFailure	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Bytes	DeliveryToAmazonOpenSearchServerless.Bytes	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.DataFreshness	DeliveryToAmazonOpenSearchServerless.DataFreshness	Histogram	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeliveryToAmazonOpenSearchServerless.DeliveryRejected	DeliveryToAmazonOpenSearchServerless.DeliveryRejected	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Records	DeliveryToAmazonOpenSearchServerless.Records	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Success	DeliveryToAmazonOpenSearchServerless.Success	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchService.AuthFailure	DeliveryToAmazonOpenSearchService.AuthFailure	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchService.Bytes	DeliveryToAmazonOpenSearchService.Bytes	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeliveryToAmazonOpenSearchService.DataFreshness	DeliveryToAmazonOpenSearchService.DataFreshness	Histogram	DeliveryStreamName
DeliveryToAmazonOpenSearchService.DeliveryRejected	DeliveryToAmazonOpenSearchService.DeliveryRejected	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchService.Records	DeliveryToAmazonOpenSearchService.Records	Sum	DeliveryStreamName
DeliveryToAmazonOpenSearchService.Success	DeliveryToAmazonOpenSearchService.Success	Sum	DeliveryStreamName
DeliveryToHttpEndpoint.Bytes	DeliveryToHttpEndpoint.Bytes	Sum	DeliveryStreamName
DeliveryToHttpEndpoint.DataFreshness	DeliveryToHttpEndpoint.DataFreshness	Histogram	DeliveryStreamName
DeliveryToHttpEndpoint.ProcessedBytes	DeliveryToHttpEndpoint.ProcessedBytes	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeliveryToHttpEndpoint.ProcessedRecords	DeliveryToHttpEndpoint.ProcessedRecords	Sum	DeliveryStreamName
DeliveryToHttpEndpoint.Records	DeliveryToHttpEndpoint.Records	Sum	DeliveryStreamName
DeliveryToHttpEndpoint.Success	DeliveryToHttpEndpoint.Success	Sum	DeliveryStreamName
DeliveryToIceberg.Bytes	DeliveryToIceberg.Bytes	Sum	DeliveryStreamName ,IcebergTableName
DeliveryToIceberg.DataFreshness	DeliveryToIceberg.DataFreshness	Histogram	DeliveryStreamName ,IcebergTableName
DeliveryToIceberg.Success	DeliveryToIceberg.Success	Sum	DeliveryStreamName ,IcebergTableName
DeliveryToIceberg.SuccessfulRowCount	DeliveryToIceberg.SuccessfulRowCount	Sum	DeliveryStreamName ,IcebergTableName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeliveryToRedshift.Bytes	DeliveryToRedshift.Bytes	Sum	DeliveryStreamName
DeliveryToRedshift.Records	DeliveryToRedshift.Records	Sum	DeliveryStreamName
DeliveryToRedshift.Success	DeliveryToRedshift.Success	Sum	DeliveryStreamName
DeliveryToS3.Bytes	DeliveryToS3.Bytes	Sum	DeliveryStreamName
DeliveryToS3.DataFreshness	DeliveryToS3.DataFreshness	Histogram	DeliveryStreamName
DeliveryToS3.Records	DeliveryToS3.Records	Sum	DeliveryStreamName
DeliveryToS3.Success	DeliveryToS3.Success	Sum	DeliveryStreamName
DeliveryToSnowflake.Bytes	DeliveryToSnowflake.Bytes	Sum	DeliveryStreamName
DeliveryToSnowflake.DataCommitLatency	DeliveryToSnowflake.DataCommitLatency	Histogram	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeliveryToSnowflake.DataFreshness	DeliveryToSnowflake.DataFreshness	Histogram	DeliveryStreamName
DeliveryToSnowflake.Records	DeliveryToSnowflake.Records	Sum	DeliveryStreamName
DeliveryToSnowflake.Success	DeliveryToSnowflake.Success	Sum	DeliveryStreamName
DeliveryToSplunk.Bytes	DeliveryToSplunk.Bytes	Sum	DeliveryStreamName
DeliveryToSplunk.DataAckLatency	DeliveryToSplunk.DataAckLatency	Histogram	DeliveryStreamName
DeliveryToSplunk.DataFreshness	DeliveryToSplunk.DataFreshness	Histogram	DeliveryStreamName
DeliveryToSplunk.Records	DeliveryToSplunk.Records	Sum	DeliveryStreamName
DeliveryToSplunk.Success	DeliveryToSplunk.Success	Sum	DeliveryStreamName
DescribeDeliveryStream.Latency	DescribeDeliveryStream.Latency	Histogram	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DescribeDeliveryStream.Requests	DescribeDeliveryStream.Requests	Sum	DeliveryStreamName
ExecuteProcessing.Duration	ExecuteProcessing.Duration	Histogram	DeliveryStreamName
ExecuteProcessing.Success	ExecuteProcessing.Success	Sum	DeliveryStreamName
FailedConversion.Bytes	FailedConversion.Bytes	Sum	DeliveryStreamName
FailedConversion.Records	FailedConversion.Records	Sum	DeliveryStreamName
FailedValidation.Bytes	FailedValidation.Bytes	Sum	DeliveryStreamName
FailedValidation.Records	FailedValidation.Records	Sum	DeliveryStreamName
IncomingBytes	IncomingBytes	Sum	DeliveryStreamName
IncomingPutRequests	IncomingPutRequests	Sum	DeliveryStreamName
IncomingRecords	IncomingRecords	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KMSKeyAccessDenied	KMSKeyAccessDenied	Sum	DeliveryStreamName
KMSKeyDisabled	KMSKeyDisabled	Sum	DeliveryStreamName
KMSKeyInvalidState	KMSKeyInvalidState	Sum	DeliveryStreamName
KMSKeyNotFound	KMSKeyNotFound	Sum	DeliveryStreamName
KafkaOffsetLag	KafkaOffsetLag	Histogram	DeliveryStreamName
KinesisMillisBehindLatest	KinesisMillisBehindLatest	Histogram	DeliveryStreamName
ListDeliveryStreams.Latency	ListDeliveryStreams.Latency	Histogram	DeliveryStreamName
ListDeliveryStreams.Requests	ListDeliveryStreams.Requests	Sum	DeliveryStreamName
OutputDecompressedBytes.Failed	OutputDecompressedBytes.Failed	Sum	DeliveryStreamName
OutputDecompressedBytes.Success	OutputDecompressedBytes.Success	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OutputDecompressedRecords.Failed	OutputDecompressedRecords.Failed	Sum	DeliveryStreamName
OutputDecompressedRecords.Success	OutputDecompressedRecords.Success	Sum	DeliveryStreamName
PutRecord.Bytes	PutRecord.Bytes	Histogram	DeliveryStreamName
PutRecord.Latency	PutRecord.Latency	Histogram	DeliveryStreamName
PutRecord.Requests	PutRecord.Requests	Sum	DeliveryStreamName
PutRecordBatch.Bytes	PutRecordBatch.Bytes	Histogram	DeliveryStreamName
PutRecordBatch.Latency	PutRecordBatch.Latency	Histogram	DeliveryStreamName
PutRecordBatch.Records	PutRecordBatch.Records	Histogram	DeliveryStreamName
PutRecordBatch.Requests	PutRecordBatch.Requests	Sum	DeliveryStreamName
PutRequestsPerSecondLimit	PutRequestsPerSecondLimit	Histogram	DeliveryStreamName
RecordsPerSecondLimit	RecordsPerSecondLimit	Histogram	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SourceThrottled.Delay	SourceThrottled.Delay	Histogram	DeliveryStreamName
SourceToDelivery.DataFreshness	SourceToDelivery.DataFreshness	Histogram	DeliveryStreamName
SucceedConversion.Bytes	SucceedConversion.Bytes	Sum	DeliveryStreamName
SucceedConversion.Records	SucceedConversion.Records	Sum	DeliveryStreamName
SucceedProcessing.Bytes	SucceedProcessing.Bytes	Sum	DeliveryStreamName
SucceedProcessing.Records	SucceedProcessing.Records	Sum	DeliveryStreamName
ThrottledDescribeStream	ThrottledDescribeStream	Sum	DeliveryStreamName
ThrottledGetRecords	ThrottledGetRecords	Sum	DeliveryStreamName
ThrottledGetShardIterator	ThrottledGetShardIterator	Sum	DeliveryStreamName
ThrottledRecords	ThrottledRecords	Sum	DeliveryStreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
UpdateDeliveryStream.Latency	UpdateDeliveryStream.Latency	Histogram	DeliveryStreamName
UpdateDeliveryStream.Requests	UpdateDeliveryStream.Requests	Sum	DeliveryStreamName

AWS/FraudDetector

The AWS/FraudDetector namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/frauddetector. All enriched instruments use Delta temporality.

- AWS::FraudDetector::Detector

AWS::FraudDetector::Detector

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GetEventPrediction	GetEventPrediction	Sum	DetectorID
GetEventPrediction4xxError	GetEventPrediction4xxError	Sum	DetectorID
GetEventPrediction5xxError	GetEventPrediction5xxError	Sum	DetectorID

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GetEventPrediction Latency	GetEventPrediction Latency	Histogram	DetectorID
ModelInvocation	ModelInvocation	Sum	DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
ModelInvocationError	ModelInvocationError	Sum	DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
ModelInvocationLatency	ModelInvocationLatency	Histogram	DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
OutcomeReturned	OutcomeReturned	Sum	DetectorID ,DetectorVersionID , OutcomeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Prediction	Prediction	Sum	DetectorID ,DetectorVersionID
PredictionError	PredictionError	Sum	DetectorID ,DetectorVersionID
PredictionLatency	PredictionLatency	Histogram	DetectorID ,DetectorVersionID
RuleEvaluateFalse	RuleEvaluateFalse	Sum	DetectorID ,DetectorVersionID ,RuleID
RuleEvaluateTrue	RuleEvaluateTrue	Sum	DetectorID ,DetectorVersionID ,RuleID
RuleNotEvaluated	RuleNotEvaluated	Sum	DetectorID ,DetectorVersionID ,RuleID
VariableUsed	VariableUsed	Sum	DetectorID ,DetectorVersionID , VariableName

AWS/GameLift

The AWS/GameLift namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/gamelift. All enriched instruments use Delta temporality.

- AWS::GameLift::Fleet

- AWS::GameLift::GameSessionQueue
- AWS::GameLift::MatchmakingConfiguration

AWS::GameLift::Fleet

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`ActivatingGameSessions`	`ActivatingGameSessions`	Histogram	`FleetId`, `Location`
`ActiveGameServerContainerGroups`	`ActiveGameServerContainerGroups`	Histogram	`FleetId`, `Location`
`ActiveGameSessions`	`ActiveGameSessions`	Histogram	`FleetId`, `Location`
`ActiveInstances`	`ActiveInstances`	Histogram	`FleetId`, `Location`
`ActiveServerProcesses`	`ActiveServerProcesses`	Histogram	`FleetId`, `Location`
`AvailableGameSessions`	`AvailableGameSessions`	Histogram	`FleetId`, `Location`
`AvailablePlayerSessions`	`AvailablePlayerSessions`	Histogram	`FleetId`, `Location`
`CPUUtilization`	`CPUUtilization`	Histogram	`FleetId`, `Location`
`ConcurrentActivata`	`ConcurrentActivata`	Histogram	`FleetId`, `Location`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
bleGameSessions	bleGameSessions		
ContainerCPUReservation	ContainerCPUReservation	Histogram	FleetId, Location
ContainerCPUUtilizationPerInstance	ContainerCPUUtilizationPerInstance	Histogram	FleetId, Location
ContainerMemoryReservation	ContainerMemoryReservation	Histogram	FleetId, Location
ContainerMemoryUtilization	ContainerMemoryUtilization	Histogram	FleetId, Location
ContainerNetworkIn	ContainerNetworkIn	Sum	ContainerGroupType, FleetId, Location
ContainerNetworkOut	ContainerNetworkOut	Sum	ContainerGroupType, FleetId, Location
ContainerStorageReadBytes	ContainerStorageReadBytes	Sum	ContainerGroupType, FleetId, Location
ContainerStorageWriteBytes	ContainerStorageWriteBytes	Sum	ContainerGroupType, FleetId, Location

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CurrentPlayerSessions	CurrentPlayerSessions	Histogram	FleetId, Location
DesiredInstances	DesiredInstances	Histogram	FleetId, Location
DiskReadBytes	DiskReadBytes	Sum	FleetId, Location
DiskReadOps	DiskReadOps	Sum	FleetId, Location
DiskWriteBytes	DiskWriteBytes	Sum	FleetId, Location
DiskWriteOps	DiskWriteOps	Sum	FleetId, Location
GameSessionInterruptions	GameSessionInterruptions	Sum	FleetId, Location
HealthyServerProcesses	HealthyServerProcesses	Histogram	FleetId, Location
IdleGameServerContainerGroups	IdleGameServerContainerGroups	Histogram	FleetId, Location
IdleInstances	IdleInstances	Histogram	FleetId, Location
InstanceCPUUtilization	InstanceCPUUtilization	Histogram	FleetId, Location
InstanceInterruptions	InstanceInterruptions	Sum	FleetId, Location
InstanceNetworkIn	InstanceNetworkIn	Sum	FleetId, Location

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InstanceNetworkOut	InstanceNetworkOut	Sum	FleetId, Location
InstanceStorageReadBytes	InstanceStorageReadBytes	Sum	FleetId, Location
InstanceStorageReadOps	InstanceStorageReadOps	Sum	FleetId, Location
InstanceStorageWriteBytes	InstanceStorageWriteBytes	Sum	FleetId, Location
InstanceStorageWriteOps	InstanceStorageWriteOps	Sum	FleetId, Location
MaxInstances	MaxInstances	Histogram	FleetId, Location
MinInstances	MinInstances	Histogram	FleetId, Location
NetworkIn	NetworkIn	Sum	FleetId, Location
NetworkOut	NetworkOut	Sum	FleetId, Location
PendingGameServerContainerGroups	PendingGameServerContainerGroups	Histogram	FleetId, Location
PendingInstances	PendingInstances	Histogram	FleetId, Location
PercentAvailableGameSessions	PercentAvailableGameSessions	Histogram	FleetId, Location

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PercentHealthyServerProcesses	PercentHealthyServerProcesses	Histogram	FleetId, Location
PercentIdleInstances	PercentIdleInstances	Histogram	FleetId, Location
PlayerGatewayBytesIn	PlayerGatewayBytesIn	Sum	FleetId
PlayerGatewayBytesOut	PlayerGatewayBytesOut	Sum	FleetId, Location
PlayerGatewayBytesThrottled	PlayerGatewayBytesThrottled	Sum	FleetId, Location
PlayerGatewayPacketsIn	PlayerGatewayPacketsIn	Sum	FleetId, Location
PlayerGatewayPacketsOut	PlayerGatewayPacketsOut	Sum	FleetId, Location
PlayerGatewayPacketsThrottled	PlayerGatewayPacketsThrottled	Sum	FleetId, Location
PlayerGatewayPlayerSessions	PlayerGatewayPlayerSessions	Sum	FleetId, Location
PlayerSessionActivations	PlayerSessionActivations	Sum	FleetId, Location

Metric	OpenTelemetry metric	Instrument type	Data point attributes
RecycledInstances	RecycledInstances	Sum	FleetId, Location
ServerProcessAbnormalTerminations	ServerProcessAbnormalTerminations	Sum	FleetId, Location
ServerProcessActivations	ServerProcessActivations	Sum	FleetId, Location
ServerProcessTerminations	ServerProcessTerminations	Sum	FleetId, Location
TerminatingGameServerContainerGroups	TerminatingGameServerContainerGroups	Histogram	FleetId, Location
TerminatingInstances	TerminatingInstances	Histogram	FleetId, Location
UnhealthyGameServerContainerGroupsReplaced	UnhealthyGameServerContainerGroupsReplaced	Sum	FleetId, Location
UnhealthyInstancesReplaced	UnhealthyInstancesReplaced	Sum	FleetId, Location

AWS::GameLift::GameSessionQueue

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`AverageWaitTime`	`AverageWaitTime`	Histogram	`Location`, `QueueName`
`FirstChoiceNotViable`	`FirstChoiceNotViable`	Sum	`QueueName`
`FirstChoiceOutOfCapacity`	`FirstChoiceOutOfCapacity`	Sum	`QueueName`
`GameSessionPlaced`	`GameSessionPlaced`	Sum	`LocationName` , `QueueName`
`LowestLatencyPlacement`	`LowestLatencyPlacement`	Sum	`QueueName`
`LowestPricePlacement`	`LowestPricePlacement`	Sum	`QueueName`
`PlacementApNortheast1`	`PlacementApNortheast1`	Sum	`QueueName`
`PlacementApNortheast2`	`PlacementApNortheast2`	Sum	`QueueName`
`PlacementApSouth1`	`PlacementApSouth1`	Sum	`QueueName`
`PlacementApSoutheast2`	`PlacementApSoutheast2`	Sum	`QueueName`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Placement EuCentral1	Placement EuCentral1	Sum	QueueName
Placement EuNorth1	Placement EuNorth1	Sum	QueueName
Placement EuWest1	Placement EuWest1	Sum	QueueName
Placement EuWest2	Placement EuWest2	Sum	QueueName
Placement MeSouth1	Placement MeSouth1	Sum	QueueName
Placement SaEast1	Placement SaEast1	Sum	QueueName
Placement UsEast1	Placement UsEast1	Sum	QueueName
Placement UsEast2	Placement UsEast2	Sum	QueueName
Placement UsWest1	Placement UsWest1	Sum	QueueName
Placement UsWest2	Placement UsWest2	Sum	QueueName
Placement sCanceled	Placement sCanceled	Sum	QueueName
Placement sStarted	Placement sStarted	Sum	QueueName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PlacementsSucceeded	PlacementsSucceeded	Sum	QueueName
PlacementsTimedOut	PlacementsTimedOut	Sum	QueueName
QueueDepth	QueueDepth	Histogram	Location, QueueName

AWS::GameLift::MatchmakingConfiguration

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CurrentTickets	CurrentTickets	Histogram	ConfigurationName
MatchesAccepted	MatchesAccepted	Sum	ConfigurationName
MatchesCreated	MatchesCreated	Sum	ConfigurationName
MatchesPlaced	MatchesPlaced	Sum	ConfigurationName
MatchesRejected	MatchesRejected	Sum	ConfigurationName
MatchmakingSearchTime	MatchmakingSearchTime	Histogram	ConfigurationName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PlayersStarted	PlayersStarted	Sum	ConfigurationName
TicketsStarted	TicketsStarted	Sum	ConfigurationName
TicketsTimedOut	TicketsTimedOut	Sum	ConfigurationName
TimeToMatch	TimeToMatch	Histogram	ConfigurationName
TimeToTicketCancel	TimeToTicketCancel	Histogram	ConfigurationName
TimeToTicketSuccess	TimeToTicketSuccess	Histogram	ConfigurationName

AWS/GatewayELB

The AWS/GatewayELB namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticloadbalancing. All enriched instruments use Delta temporality.

- AWS::ElasticLoadBalancingV2::LoadBalancer

AWS::ElasticLoadBalancingV2::LoadBalancer

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveFlowCount	ActiveFlowCount	Histogram	AvailabilityZone , LoadBalancer
HealthyHostCount	HealthyHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
NewFlowCount	NewFlowCount	Sum	AvailabilityZone , LoadBalancer
ProcessedBytes	ProcessedBytes	Sum	AvailabilityZone , LoadBalancer
UnHealthyHostCount	UnHealthyHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup

AWS/IVSChat

The AWS/IVSChat namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/ivschat. All enriched instruments use Delta temporality.

- AWS::IVSChat::LoggingConfiguration

AWS::IVSChat::LoggingConfiguration

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
LogDestinationAccessDeniedError	LogDestinationAccessDeniedError	Sum	LoggingConfiguration
LogDestinationErrors	LogDestinationErrors	Sum	LoggingConfiguration
LogDestinationResourceNotFoundErrors	LogDestinationResourceNotFoundErrors	Sum	LoggingConfiguration

AWS/IoT

The AWS/IoT namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/iot. All enriched instruments use Delta temporality.

- AWS::IoT::CACertificate
- AWS::IoT::ScheduledAudit
- AWS::IoT::SecurityProfile
- AWS::IoT::TopicRule
- job

AWS::IoT::CACertificate

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ProvisionThing.ClientError	ProvisionThing.ClientError	Sum	CaCertificateId
ProvisionThing.ServerError	ProvisionThing.ServerError	Sum	CaCertificateId
ProvisionThing.Success	ProvisionThing.Success	Sum	CaCertificateId

AWS::IoT::ScheduledAudit

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MisconfiguredDeviceDefenderNotification	MisconfiguredDeviceDefenderNotification	Sum	CheckName,ScheduledAuditName
NonCompliantResources	NonCompliantResources	Sum	CheckName,ScheduledAuditName
ResourcesEvaluated	ResourcesEvaluated	Sum	CheckName,ScheduledAuditName

AWS::IoT::SecurityProfile

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BehaviorEvaluationCompleted	BehaviorEvaluationCompleted	Sum	BehaviorName ,SecurityProfileName
BehaviorEvaluationSkipped	BehaviorEvaluationSkipped	Sum	BehaviorName ,SecurityProfileName
MisconfiguredDeviceDefenderNotification	MisconfiguredDeviceDefenderNotification	Sum	BehaviorName ,SecurityProfileName
NumOfMetricsExceedingSizeLimit	NumOfMetricsExceedingSizeLimit	Sum	BehaviorName ,SecurityProfileName
NumOfMetricsExported	NumOfMetricsExported	Sum	BehaviorName ,SecurityProfileName
NumOfMetricsSkipped	NumOfMetricsSkipped	Sum	BehaviorName ,SecurityProfileName
Violations	Violations	Sum	BehaviorName ,SecurityProfileName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ViolationsCleared	ViolationsCleared	Sum	BehaviorName ,SecurityProfileName
ViolationsInvalidated	ViolationsInvalidated	Sum	BehaviorName ,SecurityProfileName

AWS::IoT::TopicRule

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ErrorActionFailure	ErrorActionFailure	Sum	ActionType , RuleName
ErrorActionHttpDestinationNotEnabled	ErrorActionHttpDestinationNotEnabled	Sum	ActionType , RuleName
ErrorActionSuccess	ErrorActionSuccess	Sum	ActionType , RuleName
Failure	Failure	Sum	ActionType , RuleName
HttpCode_4XX	HttpCode_4XX	Sum	ActionType , RuleName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HttpCode_5XX	HttpCode_5XX	Sum	ActionType , RuleName
HttpCode_Other	HttpCode_Other	Sum	ActionType , RuleName
HttpInvalidUrl	HttpInvalidUrl	Sum	ActionType , RuleName
HttpRequestTimeout	HttpRequestTimeout	Sum	ActionType , RuleName
HttpUnknownHost	HttpUnknownHost	Sum	ActionType , RuleName
ParseError	ParseError	Sum	RuleName
RequestTimeout	RequestTimeout	Sum	ActionType , RuleName
RuleMessageThrottled	RuleMessageThrottled	Sum	RuleName
RuleNotFound	RuleNotFound	Sum	RuleName
RulesExecuted	RulesExecuted	Sum	RuleName
SaslAuthFailed	SaslAuthFailed	Sum	ActionType , RuleName
Success	Success	Sum	ActionType , RuleName
TopicMatch	TopicMatch	Sum	RuleName

job

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`CanceledJobExecutionCount`	`CanceledJobExecutionCount`	Sum	`JobId`
`CanceledJobExecutionTotalCount`	`CanceledJobExecutionTotalCount`	Histogram	`JobId`
`ClientErrorCount`	`ClientErrorCount`	Sum	`JobId`
`FailedJobExecutionCount`	`FailedJobExecutionCount`	Sum	`JobId`
`FailedJobExecutionTotalCount`	`FailedJobExecutionTotalCount`	Histogram	`JobId`
`InProgressJobExecutionCount`	`InProgressJobExecutionCount`	Sum	`JobId`
`InProgressJobExecutionTotalCount`	`InProgressJobExecutionTotalCount`	Histogram	`JobId`
`QueuedJobExecutionCount`	`QueuedJobExecutionCount`	Sum	`JobId`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
QueuedJobExecutionTotalCount	QueuedJobExecutionTotalCount	Histogram	JobId
RejectedJobExecutionCount	RejectedJobExecutionCount	Sum	JobId
RejectedJobExecutionTotalCount	RejectedJobExecutionTotalCount	Histogram	JobId
RemovedJobExecutionCount	RemovedJobExecutionCount	Sum	JobId
RemovedJobExecutionTotalCount	RemovedJobExecutionTotalCount	Histogram	JobId
ServerErrorCount	ServerErrorCount	Sum	JobId
SucceededJobExecutionCount	SucceededJobExecutionCount	Sum	JobId
SucceededJobExecutionTotalCount	SucceededJobExecutionTotalCount	Histogram	JobId
SucceededJobExecutionCount	SucceededJobExecutionCount	Sum	JobId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SucceededJobExecutionTotalCount	SucceededJobExecutionTotalCount	Histogram	JobId
TimedOutJobExecutionTotalCount	TimedOutJobExecutionTotalCount	Histogram	JobId

AWS/KMS

The AWS/KMS namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/kms. All enriched instruments use Delta temporality.

- AWS::KMS::Key

AWS::KMS::Key

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SecondsUntilKeyMaterialExpiration	SecondsUntilKeyMaterialExpiration	Histogram	KeyId

AWS/Kafka

The AWS/Kafka namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/kafka. All enriched instruments use Delta temporality.

- broker
- broker-client-auth
- cluster
- consumer-group
- consumer-group-partition
- replicator
- replicator-topic
- topic

broker

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BurstBalance	BurstBalance	Histogram	Broker ID, Cluster Name
BwInAllowanceExceeded	BwInAllowanceExceeded	Sum	Broker ID, Cluster Name
BwOutAllowanceExceeded	BwOutAllowanceExceeded	Sum	Broker ID, Cluster Name
CPUCreditBalance	CPUCreditBalance	Histogram	Broker ID, Cluster Name
CPUCreditUsage	CPUCreditUsage	Sum	Broker ID, Cluster Name
ConnectionCloseRate	ConnectionCloseRate	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConnectionCount	ConnectionCount	Histogram	Broker ID, Cluster Name
ConnectionCreationRate	ConnectionCreationRate	Histogram	Broker ID, Cluster Name
ConntrackAllowanceExceeded	ConntrackAllowanceExceeded	Sum	Broker ID, Cluster Name
CpuIdle	CpuIdle	Histogram	Broker ID, Cluster Name
CpuIoWait	CpuIoWait	Histogram	Broker ID, Cluster Name
CpuSystem	CpuSystem	Histogram	Broker ID, Cluster Name
CpuUser	CpuUser	Histogram	Broker ID, Cluster Name
FetchConsumerLocalTimeMsMean	FetchConsumerLocalTimeMsMean	Histogram	Broker ID, Cluster Name
FetchConsumerRequestQueueTimeMsMean	FetchConsumerRequestQueueTimeMsMean	Histogram	Broker ID, Cluster Name
FetchConsumerResponseQueueTimeMsMean	FetchConsumerResponseQueueTimeMsMean	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FetchConsumerResponseSendTimeMsMean	FetchConsumerResponseSendTimeMsMean	Histogram	Broker ID, Cluster Name
FetchConsumerTotalTimeMsMean	FetchConsumerTotalTimeMsMean	Histogram	Broker ID, Cluster Name
FetchFollowerLocalTimeMsMean	FetchFollowerLocalTimeMsMean	Histogram	Broker ID, Cluster Name
FetchFollowerRequestQueueTimeMsMean	FetchFollowerRequestQueueTimeMsMean	Histogram	Broker ID, Cluster Name
FetchFollowerResponseQueueTimeMsMean	FetchFollowerResponseQueueTimeMsMean	Histogram	Broker ID, Cluster Name
FetchFollowerResponseSendTimeMsMean	FetchFollowerResponseSendTimeMsMean	Histogram	Broker ID, Cluster Name
FetchFollowerTotalTimeMsMean	FetchFollowerTotalTimeMsMean	Histogram	Broker ID, Cluster Name
FetchMessageConversionsPerSec	FetchMessageConversionsPerSec	Sum	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FetchThrottleByteRate	FetchThrottleByteRate	Sum	Broker ID, Cluster Name
FetchThrottleQueueSize	FetchThrottleQueueSize	Histogram	Broker ID, Cluster Name
FetchThrottleTime	FetchThrottleTime	Histogram	Broker ID, Cluster Name
HeapMemoryAfterGC	HeapMemoryAfterGC	Histogram	Broker ID, Cluster Name
IAMNumberOfConnectionRequests	IAMNumberOfConnectionRequests	Sum	Broker ID, Cluster Name
IAMTooManyConnections	IAMTooManyConnections	Sum	Broker ID, Cluster Name
KafkaAppLogsDiskUsed	KafkaAppLogsDiskUsed	Histogram	Broker ID, Cluster Name
KafkaDataLogsDiskUsed	KafkaDataLogsDiskUsed	Histogram	Broker ID, Cluster Name
LeaderCount	LeaderCount	Histogram	Broker ID, Cluster Name
LinklocalAllowanceExceeded	LinklocalAllowanceExceeded	Sum	Broker ID, Cluster Name
MemoryBuffered	MemoryBuffered	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MemoryCached	MemoryCached	Histogram	Broker ID, Cluster Name
MemoryFree	MemoryFree	Histogram	Broker ID, Cluster Name
MemoryUsed	MemoryUsed	Histogram	Broker ID, Cluster Name
MessagesInPerSec	MessagesInPerSec	Sum	Broker ID, Cluster Name
NetworkProcessorAvgIdlePercent	NetworkProcessorAvgIdlePercent	Histogram	Broker ID, Cluster Name
NetworkRxDropped	NetworkRxDropped	Sum	Broker ID, Cluster Name
NetworkRxErrors	NetworkRxErrors	Sum	Broker ID, Cluster Name
NetworkRxPackets	NetworkRxPackets	Sum	Broker ID, Cluster Name
NetworkTxDropped	NetworkTxDropped	Sum	Broker ID, Cluster Name
NetworkTxErrors	NetworkTxErrors	Sum	Broker ID, Cluster Name
NetworkTxPackets	NetworkTxPackets	Sum	Broker ID, Cluster Name
PartitionCount	PartitionCount	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PpsAllowanceExceeded	PpsAllowanceExceeded	Sum	Broker ID, Cluster Name
ProduceLocalTimeMsMean	ProduceLocalTimeMsMean	Histogram	Broker ID, Cluster Name
ProduceMessageConversionsPerSec	ProduceMessageConversionsPerSec	Sum	Broker ID, Cluster Name
ProduceMessageConversionsTimeMsMean	ProduceMessageConversionsTimeMsMean	Histogram	Broker ID, Cluster Name
ProduceRequestQueueTimeMsMean	ProduceRequestQueueTimeMsMean	Histogram	Broker ID, Cluster Name
ProduceResponseQueueTimeMsMean	ProduceResponseQueueTimeMsMean	Histogram	Broker ID, Cluster Name
ProduceResponseSendTimeMsMean	ProduceResponseSendTimeMsMean	Histogram	Broker ID, Cluster Name
ProduceThrottleByteRate	ProduceThrottleByteRate	Sum	Broker ID, Cluster Name
ProduceThrottleQueueSize	ProduceThrottleQueueSize	Histogram	Broker ID, Cluster Name
ProduceThrottleTime	ProduceThrottleTime	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ProduceTotalTimeMsMean	ProduceTotalTimeMsMean	Histogram	Broker ID, Cluster Name
RemoteCopyBytesPerSec	RemoteCopyBytesPerSec	Sum	Broker ID, Cluster Name
RemoteCopyErrorsPerSec	RemoteCopyErrorsPerSec	Sum	Broker ID, Cluster Name
RemoteCopyLagBytes	RemoteCopyLagBytes	Histogram	Broker ID, Cluster Name
RemoteFetchBytesPerSec	RemoteFetchBytesPerSec	Sum	Broker ID, Cluster Name
RemoteFetchErrorsPerSec	RemoteFetchErrorsPerSec	Sum	Broker ID, Cluster Name
RemoteFetchRequestsPerSec	RemoteFetchRequestsPerSec	Sum	Broker ID, Cluster Name
RemoteLogManagerTasksAvgIdlePercent	RemoteLogManagerTasksAvgIdlePercent	Histogram	Broker ID, Cluster Name
RemoteLogReaderAvgIdlePercent	RemoteLogReaderAvgIdlePercent	Histogram	Broker ID, Cluster Name
RemoteLogReaderTaskQueueSize	RemoteLogReaderTaskQueueSize	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
RemoteLogSizeBytes	RemoteLogSizeBytes	Histogram	Broker ID, Cluster Name
ReplicationBytesInPerSec	ReplicationBytesInPerSec	Sum	Broker ID, Cluster Name
ReplicationBytesOutPerSec	ReplicationBytesOutPerSec	Sum	Broker ID, Cluster Name
RequestBytesMean	RequestBytesMean	Histogram	Broker ID, Cluster Name
RequestExemptFromThrottleTime	RequestExemptFromThrottleTime	Histogram	Broker ID, Cluster Name
RequestHandlerAvgIdlePercent	RequestHandlerAvgIdlePercent	Histogram	Broker ID, Cluster Name
RequestThrottleQueueSize	RequestThrottleQueueSize	Histogram	Broker ID, Cluster Name
RequestThrottleTime	RequestThrottleTime	Histogram	Broker ID, Cluster Name
RequestTime	RequestTime	Histogram	Broker ID, Cluster Name
RootDiskUsed	RootDiskUsed	Histogram	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SwapFree	SwapFree	Histogram	Broker ID, Cluster Name
SwapUsed	SwapUsed	Histogram	Broker ID, Cluster Name
TCPConnections	TCPConnections	Sum	Broker ID, Cluster Name
TrafficBytes	TrafficBytes	Sum	Broker ID, Cluster Name
TrafficShaping	TrafficShaping	Sum	Broker ID, Cluster Name
UnderMinIsrPartitionCount	UnderMinIsrPartitionCount	Histogram	Broker ID, Cluster Name
UnderReplicatedPartitions	UnderReplicatedPartitions	Histogram	Broker ID, Cluster Name
UserPartitionExists	UserPartitionExists	Histogram	Broker ID, Cluster Name
VolumeQueueLength	VolumeQueueLength	Histogram	Broker ID, Cluster Name
VolumeReadBytes	VolumeReadBytes	Sum	Broker ID, Cluster Name
VolumeReadOps	VolumeReadOps	Sum	Broker ID, Cluster Name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
VolumeTotalReadTime	VolumeTotalReadTime	Sum	Broker ID, Cluster Name
VolumeTotalWriteTime	VolumeTotalWriteTime	Sum	Broker ID, Cluster Name
VolumeWriteBytes	VolumeWriteBytes	Sum	Broker ID, Cluster Name
VolumeWriteOps	VolumeWriteOps	Sum	Broker ID, Cluster Name
ZooKeeperRequestLatencyMsMean	ZooKeeperRequestLatencyMsMean	Histogram	Broker ID, Cluster Name
ZooKeeperSessionState	ZooKeeperSessionState	Histogram	Broker ID, Cluster Name

broker-client-auth

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ClientConnectionCount	ClientConnectionCount	Histogram	Broker ID, Client Authentication , Cluster Name

cluster

- `cloud.resource_id` enrichment: Not supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveControllerCount	ActiveControllerCount	Histogram	Cluster Name
GlobalPartitionCount	GlobalPartitionCount	Histogram	Cluster Name
GlobalTopicCount	GlobalTopicCount	Histogram	Cluster Name
OfflinePartitionsCount	OfflinePartitionsCount	Histogram	Cluster Name

consumer-group

- `cloud.resource_id` enrichment: Not supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EstimatedMaxTimeLag	EstimatedMaxTimeLag	Histogram	Cluster Name, Consumer Group, Topic
MaxOffsetLag	MaxOffsetLag	Histogram	Cluster Name, Consumer Group, Topic

Metric	OpenTelemetry metric	Instrument type	Data point attributes
RollingEstimatedTimeLagMax	RollingEstimatedTimeLagMax	Histogram	Cluster Name, Consumer Group, Topic
SumOffsetLag	SumOffsetLag	Histogram	Cluster Name, Consumer Group, Topic

consumer-group-partition

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
EstimatedTimeLag	EstimatedTimeLag	Histogram	Consumer Group, Partition , Topic
OffsetLag	OffsetLag	Histogram	Consumer Group, Partition , Topic
RollingEstimatedTimeLag	RollingEstimatedTimeLag	Histogram	Consumer Group, Partition , Topic

replicator

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConsumerGroupCount	ConsumerGroupCount	Histogram	ReplicatorName
ConsumerGroupOffsetSyncFailure	ConsumerGroupOffsetSyncFailure	Sum	ReplicatorName
MessageLag	MessageLag	Histogram	ReplicatorName
ReplicationLatency	ReplicationLatency	Histogram	ReplicatorName
ReplicatorBytesInPerSec	ReplicatorBytesInPerSec	Sum	ReplicatorName
ReplicatorBytesOutPerSec	ReplicatorBytesOutPerSec	Sum	ReplicatorName
ReplicatorFailure	ReplicatorFailure	Sum	ReplicatorName
ReplicatorThroughput	ReplicatorThroughput	Sum	ReplicatorName
TopicCount	TopicCount	Histogram	ReplicatorName

replicator-topic

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MessageLag	MessageLagPerTopic	Histogram	ReplicatorName , Topic
ReplicationLatency	ReplicationLatencyPerTopic	Histogram	ReplicatorName , Topic
ReplicatorThroughput	ReplicatorThroughputPerTopic	Sum	ReplicatorName , Topic

topic

- cloud.resource_id enrichment: Not supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesInPerSec	BytesInPerSec	Sum	Broker ID, Cluster Name, Topic
BytesOutPerSec	BytesOutPerSec	Sum	Broker ID, Cluster Name, Topic
FetchMessageConversionsPerSec	FetchMessageConversionsPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MessagesInPerSec	MessagesInPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
ProduceMessageConversionsPerSec	ProduceMessageConversionsPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteCopyBytesPerSec	RemoteCopyBytesPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteCopyErrorsPerSec	RemoteCopyErrorsPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteFetchBytesPerSec	RemoteFetchBytesPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteFetchErrorsPerSec	RemoteFetchErrorsPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteFetchRequestsPerSec	RemoteFetchRequestsPerSecPerTopic	Sum	Broker ID, Cluster Name, Topic
RemoteLogSizeBytes	RemoteLogSizeBytesPerTopic	Histogram	Broker ID, Cluster Name, Topic

AWS/Kendra

The `AWS/Kendra` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/kendra`. All enriched instruments use Delta temporality.

- AWS::Kendra::DataSource
- AWS::Kendra::Index

AWS::Kendra::DataSource

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`DataSourceDocumentCount`	`DataSourceDocumentCount`	Histogram	`DataSourceId`, `IndexId`
`DataSourceSyncRuntime`	`DataSourceSyncRuntime`	Histogram	`DataSourceId`, `IndexId`
`DocumentsCrawled`	`DocumentsCrawled`	Sum	`DataSourceId`, `IndexId`
`DocumentsFailedToIndex`	`DocumentsFailedToIndex`	Sum	`DataSourceId`, `IndexId`
`DocumentsIndexed`	`DocumentsIndexed`	Sum	`DataSourceId`, `IndexId`
`DocumentsSkippedInvalidMetadata`	`DocumentsSkippedInvalidMetadata`	Sum	`DataSourceId`, `IndexId`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Documents SkippedNo Change	Documents SkippedNo Change	Sum	DataSourceId , IndexId
Documents Submitted ForDeletion	Documents Submitted ForDeletion	Sum	DataSourceId , IndexId
Documents Submitted ForDeleti onFailed	Documents Submitted ForDeleti onFailed	Sum	DataSourceId , IndexId
Documents Submitted ForIndexing	Documents Submitted ForIndexing	Sum	DataSourceId , IndexId
Documents Submitted ForIndexi ngFailed	Documents Submitted ForIndexi ngFailed	Sum	DataSourceId , IndexId
Documents WithoutAcl	Documents WithoutAcl	Sum	AwsAccountId , DataSourceId , DataSourceType , IndexId
MetadataF ilesCrawled	MetadataF ilesCrawled	Sum	DataSourceId , IndexId
Principal GroupsScanned	Principal GroupsScanned	Sum	DataSourceId , IndexId

AWS::Kendra::Index

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ClickFeedbackSubmittedCount	ClickFeedbackSubmittedCount	Sum	IndexId
GroupProcessed	GroupProcessed	Sum	IndexId
GroupProcessedButNotPersistent	GroupProcessedButNotPersistent	Sum	IndexId
GroupReceived	GroupReceived	Sum	IndexId
GroupUpdateLatencyInMilliSeconds	GroupUpdateLatencyInMilliSeconds	Histogram	IndexId
GroupWithOlderOrderingIdSkipped	GroupWithOlderOrderingIdSkipped	Sum	IndexId
IndexDocumentCount	IndexDocumentCount	Histogram	IndexId
IndexDocumentStorageSize	IndexDocumentStorageSize	Histogram	IndexId
IndexQueryCount	IndexQueryCount	Sum	IndexId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ProvisionedIndexDocumentCount	ProvisionedIndexDocumentCount	Histogram	IndexId
ProvisionedIndexStorageSize	ProvisionedIndexStorageSize	Histogram	IndexId
RelevantFeedbackSubmittedCount	RelevantFeedbackSubmittedCount	Sum	IndexId

AWS/Kinesis

The AWS/Kinesis namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/kinesis. All enriched instruments use Delta temporality.

- AWS::Kinesis::Stream

AWS::Kinesis::Stream

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
GetRecords.Bytes	GetRecords.Bytes	Sum	StreamName
GetRecords.Iterato	GetRecords.Iterato	Histogram	StreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
rAgeMilli seconds	rAgeMilli seconds		
GetRecord s.Latency	GetRecord s.Latency	Histogram	StreamName
GetRecord s.Records	GetRecord s.Records	Sum	StreamName
GetRecord s.Success	GetRecord s.Success	Sum	StreamName
IncomingBytes	IncomingBytes	Sum	StreamName
IncomingRecords	IncomingRecords	Sum	StreamName
IteratorA geMilliseconds	IteratorA geMilliseconds	Histogram	ShardId, StreamName
OutgoingBytes	OutgoingBytes	Sum	ShardId, StreamName
OutgoingRecords	OutgoingRecords	Sum	ShardId, StreamName
PutRecord .Success	PutRecord .Success	Sum	StreamName
PutRecord s.Bytes	PutRecord s.Bytes	Sum	StreamName
PutRecord s.Success	PutRecord s.Success	Sum	StreamName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReadProvisionedThroughputExceeded	ReadProvisionedThroughputExceeded	Sum	StreamName
SubscribeToShard.RateExceeded	SubscribeToShard.RateExceeded	Sum	ConsumerName , StreamName
SubscribeToShard.Success	SubscribeToShard.Success	Sum	ConsumerName , StreamName
SubscribeToShardEvent.Bytes	SubscribeToShardEvent.Bytes	Sum	ConsumerName , StreamName
SubscribeToShardEvent.MillisBehindLatest	SubscribeToShardEvent.MillisBehindLatest	Histogram	ConsumerName , StreamName
SubscribeToShardEvent.Records	SubscribeToShardEvent.Records	Sum	ConsumerName , StreamName
SubscribeToShardEvent.Success	SubscribeToShardEvent.Success	Sum	ConsumerName , StreamName
WriteProvisionedThroughputExceeded	WriteProvisionedThroughputExceeded	Sum	StreamName

AWS/KinesisAnalytics

The `AWS/KinesisAnalytics` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/kinesisanalytics`. All enriched instruments use Delta temporality.

- AWS::KinesisAnalyticsV2::Application

AWS::KinesisAnalyticsV2::Application

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
KPUs	KPUs	Histogram	Application
KPUs-Interactive	KPUs-Interactive	Histogram	Application
backPressuredTimeMsPerSecond	backPressuredTimeMsPerSecond	Histogram	Application
busyTimeMsPerSecond	busyTimeMsPerSecond	Histogram	Application
bytesRequestedPerFetch	bytesRequestedPerFetch	Histogram	Application, Flow, Id
committedOffsets	committedOffsets	Histogram	Application, Flow, Topic
containerCPUUtilization	containerCPUUtilization	Histogram	Application

Metric	OpenTelemetry metric	Instrument type	Data point attributes
container DiskUtilization	container DiskUtilization	Histogram	Application
container MemoryUtilization	container MemoryUtilization	Histogram	Application
cpuUtilization	cpuUtilization	Histogram	Application
currentInputWatermark	currentInputWatermark	Histogram	Application
currentOffsets	currentOffsets	Histogram	Application , Flow, Topic
currentOutputWatermark	currentOutputWatermark	Histogram	Application
downtime	downtime	Histogram	Application
fullRestarts	fullRestarts	Sum	Application
heapMemoryUtilization	heapMemoryUtilization	Histogram	Application
idleTimeMsPerSecond	idleTimeMsPerSecond	Histogram	Application
jobmanager rFileDescriptorsMax	jobmanager rFileDescriptorsMax	Histogram	Application
jobmanager rFileDescriptorsOpen	jobmanager rFileDescriptorsOpen	Histogram	Application

Metric	OpenTelemetry metric	Instrument type	Data point attributes
jobmanagerHeapMemoryUtilization	jobmanagerHeapMemoryUtilization	Histogram	Application
jobmanagerMetaspaceMemoryUtilization	jobmanagerMetaspaceMemoryUtilization	Histogram	Application
lastCheckpointDuration	lastCheckpointDuration	Histogram	Application
lastCheckpointSize	lastCheckpointSize	Histogram	Application
managedMemoryTotal	managedMemoryTotal	Histogram	Application
managedMemoryUsed	managedMemoryUsed	Histogram	Application
managedMemoryUtilization	managedMemoryUtilization	Histogram	Application
metaspaceMemoryUtilization	metaspaceMemoryUtilization	Histogram	Application
millisBehindLatest	millisBehindLatest	Histogram	Application , Flow, Id
numLateRecordsDropped	numLateRecordsDropped	Sum	Application
numRecordsIn	numRecordsIn	Sum	Application

Metric	OpenTelemetry metric	Instrument type	Data point attributes
numRecordsInPerSecond	numRecordsInPerSecond	Histogram	Application
numRecordsOut	numRecordsOut	Sum	Application
numRecordsOutPerSecond	numRecordsOutPerSecond	Histogram	Application
numRestarts	numRestarts	Sum	Application
numberOfFailedCheckpoints	numberOfFailedCheckpoints	Sum	Application
oldGenerationGCCount	oldGenerationGCCount	Sum	Application
oldGenerationGCTime	oldGenerationGCTime	Sum	Application
taskmanagerFileDescriptorsMax	taskmanagerFileDescriptorsMax	Histogram	Application
taskmanagerFileDescriptorsOpen	taskmanagerFileDescriptorsOpen	Histogram	Application
threadCount	threadCount	Histogram	Application
threadsCount	threadsCount	Histogram	Application
uptime	uptime	Histogram	Application
zeppelinCpuUtilization	zeppelinCpuUtilization	Histogram	Application

Metric	OpenTelemetry metric	Instrument type	Data point attributes
zeppelinHeapMemoryUtilization	zeppelinHeapMemoryUtilization	Histogram	Application
zeppelinServerUptime	zeppelinServerUptime	Histogram	Application
zeppelinThreadCount	zeppelinThreadCount	Histogram	Application
zeppelinWaitingJobs	zeppelinWaitingJobs	Histogram	Application

AWS/Lambda

The AWS/Lambda namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/lambda. All enriched instruments use Delta temporality.

- AWS::Lambda::CapacityProvider
- AWS::Lambda::EventSourceMapping
- AWS::Lambda::Function

AWS::Lambda::CapacityProvider

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ExecutionEnvironme	ExecutionEnvironme	Histogram	CapacityProviderName ,

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ntCPUUtilization	ntCPUUtilization		FunctionName, Resource
ExecutionEnvironmentConcurrency	ExecutionEnvironmentConcurrency	Histogram	CapacityProviderName, FunctionName, Resource
ExecutionEnvironmentConcurrencyLimit	ExecutionEnvironmentConcurrencyLimit	Histogram	CapacityProviderName, FunctionName, Resource
ExecutionEnvironmentMemoryUtilization	ExecutionEnvironmentMemoryUtilization	Histogram	CapacityProviderName, FunctionName, Resource

AWS::Lambda::EventSourceMapping

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CommittedEventCount	CommittedEventCount	Sum	EventSourceMappingUUID
DeletedEventCount	DeletedEventCount	Sum	EventSourceMappingUUID

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DroppedEventCount	DroppedEventCount	Sum	EventSourceMappingUUID
EventPollerThroughputInBytes	EventPollerThroughputInBytes	Sum	EventSourceMappingUUID
EventPollerUnit	EventPollerUnit	Histogram	EventSourceMappingUUID
FailedInvokeEventCount	FailedInvokeEventCount	Sum	EventSourceMappingUUID
FilteredOutEventCount	FilteredOutEventCount	Sum	EventSourceMappingUUID
InvokedEventCount	InvokedEventCount	Sum	EventSourceMappingUUID
IteratorAge	IteratorAge	Histogram	EventSourceMappingUUID
MaxOffsetLag	MaxOffsetLag	Histogram	EventSourceMappingUUID
OnFailureDestinationDeliveredEventCount	OnFailureDestinationDeliveredEventCount	Sum	EventSourceMappingUUID
PolledEventCount	PolledEventCount	Sum	EventSourceMappingUUID
ProvisionedPollers	ProvisionedPollers	Histogram	EventSourceMappingUUID

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SumOffsetLag	SumOffsetLag	Histogram	EventSourceMappingUUID

AWS::Lambda::Function

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AsyncEventAge	AsyncEventAge	Histogram	FunctionName , Resource
AsyncEventsDropped	AsyncEventsDropped	Sum	FunctionName , Resource
AsyncEventsReceived	AsyncEventsReceived	Sum	FunctionName , Resource
CPUThrottles	CPUThrottles	Sum	FunctionName , Resource
ConcurrencyThrottles	ConcurrencyThrottles	Sum	FunctionName , Resource
ConcurrentExecutions	ConcurrentExecutions	Histogram	FunctionName , Resource
DeadLetterErrors	DeadLetterErrors	Sum	FunctionName , Resource

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DestinationDeliveryFailures	DestinationDeliveryFailures	Sum	FunctionName , Resource
DiskThrottles	DiskThrottles	Sum	FunctionName , Resource
DurableExecutionDuration	DurableExecutionDuration	Histogram	FunctionName , Resource
DurableExecutionFailed	DurableExecutionFailed	Sum	FunctionName , Resource
DurableExecutionOperations	DurableExecutionOperations	Sum	FunctionName , Resource
DurableExecutionStarted	DurableExecutionStarted	Sum	FunctionName , Resource
DurableExecutionStopped	DurableExecutionStopped	Sum	FunctionName , Resource
DurableExecutionStorageWrittenBytes	DurableExecutionStorageWrittenBytes	Sum	FunctionName , Resource
DurableExecutionSucceeded	DurableExecutionSucceeded	Sum	FunctionName , Resource
DurableExecutionTimedOut	DurableExecutionTimedOut	Sum	FunctionName , Resource

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Duration	Duration	Histogram	FunctionName , Resource
Errors	Errors	Sum	FunctionName , Resource
Invocations	Invocations	Sum	FunctionName , Resource
IteratorAge	IteratorAge	Histogram	FunctionName , Resource
MemoryThrottles	MemoryThrottles	Sum	FunctionName , Resource
OffsetLag	OffsetLag	Histogram	FunctionName , Resource
PostRuntimeExtensionsDuration	PostRuntimeExtensionsDuration	Histogram	FunctionName , Resource
ProvisionedConcurrencyInvocations	ProvisionedConcurrencyInvocations	Sum	FunctionName , Resource
ProvisionedConcurrencySpilloverInvocations	ProvisionedConcurrencySpilloverInvocations	Sum	FunctionName , Resource

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ProvisionedConcurrencyUtilization	ProvisionedConcurrencyUtilization	Histogram	FunctionName , Resource
ProvisionedConcurrentExecutions	ProvisionedConcurrentExecutions	Histogram	FunctionName , Resource
RecursiveInvocationsDropped	RecursiveInvocationsDropped	Sum	FunctionName , Resource
SignatureValidationErrors	SignatureValidationErrors	Sum	FunctionName , Resource
StreamedOutboundBytes	StreamedOutboundBytes	Sum	FunctionName , Resource
StreamedOutboundThroughput	StreamedOutboundThroughput	Histogram	FunctionName , Resource
Throttles	Throttles	Sum	FunctionName , Resource
TimeToFirstByteLatency	TimeToFirstByteLatency	Histogram	FunctionName , Resource
TimeToLastByteLatency	TimeToLastByteLatency	Histogram	FunctionName , Resource
Url4xxCount	Url4xxCount	Sum	FunctionName , Resource

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Url5xxCount	Url5xxCount	Sum	FunctionName , Resource
UrlRequestCount	UrlRequestCount	Sum	FunctionName , Resource
UrlRequestLatency	UrlRequestLatency	Histogram	FunctionName , Resource

AWS/M2

The AWS/M2 namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/m2. All enriched instruments use Delta temporality.

- AWS::M2::Application

AWS::M2::Application

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BatchJobCompletedCount	BatchJobCompletedCount	Sum	ApplicationId
BatchJobFailedCount	BatchJobFailedCount	Sum	ApplicationId
JvmMemoryFree	JvmMemoryFree	Histogram	ApplicationId
JvmMemoryMax	JvmMemoryMax	Histogram	ApplicationId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
JvmMemoryUsed	JvmMemoryUsed	Histogram	ApplicationId
Processes ActiveCount	Processes ActiveCount	Sum	ApplicationId
SessionCount	SessionCount	Histogram	ApplicationId
SharedMem oryFree	SharedMem oryFree	Histogram	ApplicationId
SharedMem oryTotal	SharedMem oryTotal	Histogram	ApplicationId
ThreadAct iveCount	ThreadAct iveCount	Histogram	ApplicationId
Transacti onComplet edCount	Transacti onComplet edCount	Sum	ApplicationId
Transacti onFailedCount	Transacti onFailedCount	Sum	ApplicationId
Transacti onResponseTime	Transacti onResponseTime	Histogram	ApplicationId

AWS/MWAA

The AWS/MWAA namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/airflow. All enriched instruments use Delta temporality.

- AWS::MWAA::Environment

AWS::MWAA::Environment

- cloud.resource_id enrichment: Supported

- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveConnectionCount	ActiveConnectionCount	Histogram	Environment
ApproximateAgeOfOldestTask	ApproximateAgeOfOldestTask	Histogram	Environment
CPUUtilization	CPUUtilization	Histogram	Cluster, Environment
CPUUtilization	CPUUtilizationPerDatabaseRole	Histogram	DatabaseRole , Environment
DatabaseConnections	DatabaseConnections	Histogram	DatabaseRole , Environment
DiskQueueDepth	DiskQueueDepth	Histogram	DatabaseRole , Environment
EphemeralStorageReserved	EphemeralStorageReserved	Histogram	Cluster, Environment
EphemeralStorageUtilized	EphemeralStorageUtilized	Histogram	Cluster, Environment
FreeableMemory	FreeableMemory	Histogram	DatabaseRole , Environment
MemoryUtilization	MemoryUtilization	Histogram	Cluster, Environment

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkReceiveThroughput	NetworkReceiveThroughput	Histogram	DatabaseRole , Environment
NetworkRxBytes	NetworkRxBytes	Sum	Cluster, Environment
NetworkTransmitThroughput	NetworkTransmitThroughput	Histogram	DatabaseRole , Environment
NetworkTxBytes	NetworkTxBytes	Sum	Cluster, Environment
QueuedTasks	QueuedTasks	Histogram	Environment
ReadIOPS	ReadIOPS	Histogram	DatabaseRole , Environment
ReadLatency	ReadLatency	Histogram	DatabaseRole , Environment
ReadThroughput	ReadThroughput	Histogram	DatabaseRole , Environment
RunningTasks	RunningTasks	Histogram	Environment
WriteIOPS	WriteIOPS	Histogram	DatabaseRole , Environment
WriteLatency	WriteLatency	Histogram	DatabaseRole , Environment
WriteThroughput	WriteThroughput	Histogram	DatabaseRole , Environment

AWS/MediaTailor

The `AWS/MediaTailor` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/mediatailor`. All enriched instruments use Delta temporality.

- AWS::MediaTailor::Channel

AWS::MediaTailor::Channel

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`4xxErrorCount`	`4xxErrorCount`	Sum	`Channel`, `Output`
`5xxErrorCount`	`5xxErrorCount`	Sum	`Channel`, `Output`
`RequestCount`	`RequestCount`	Sum	`Channel`, `Output`
`TotalTime`	`TotalTime`	Histogram	`Channel`, `Output`

AWS/MemoryDB

The `AWS/MemoryDB` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/memorydb`. All enriched instruments use Delta temporality.

- AWS::MemoryDB::Cluster

AWS::MemoryDB::Cluster

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveDefragHits	ActiveDefragHits	Sum	ClusterName , NodeName
AuthenticationFailures	AuthenticationFailures	Sum	ClusterName , NodeName
BytesReadFromDisk	BytesReadFromDisk	Sum	ClusterName , NodeName
BytesUsedForMemoryDB	BytesUsedForMemoryDB	Histogram	ClusterName , NodeName
BytesWrittenToDisk	BytesWrittenToDisk	Sum	ClusterName , NodeName
CPUUtilization	CPUUtilization	Histogram	ClusterName , NodeName
ChannelAuthorizationFailures	ChannelAuthorizationFailures	Sum	ClusterName , NodeName
CommandAuthorizationFailures	CommandAuthorizationFailures	Sum	ClusterName , NodeName
CurrConnections	CurrConnections	Histogram	ClusterName , NodeName
CurrItems	CurrItems	Histogram	ClusterName , NodeName
DB0AverageTTL	DB0AverageTTL	Histogram	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DatabaseCapacityUsagePercentage	DatabaseCapacityUsagePercentage	Histogram	ClusterName , NodeName
DatabaseMemoryUsagePercentage	DatabaseMemoryUsagePercentage	Histogram	ClusterName , NodeName
EngineCPUUtilization	EngineCPUUtilization	Histogram	ClusterName , NodeName
ErrorCount	ErrorCount	Sum	ClusterName , NodeName
EvalBasedCmds	EvalBasedCmds	Sum	ClusterName , NodeName
Evictions	Evictions	Sum	ClusterName , NodeName
FreeableMemory	FreeableMemory	Histogram	ClusterName , NodeName
GetTypeCmds	GetTypeCmds	Sum	ClusterName , NodeName
HashBasedCmds	HashBasedCmds	Sum	ClusterName , NodeName
IamAuthenticationExpirations	IamAuthenticationExpirations	Sum	ClusterName , NodeName
IamAuthenticationThrottling	IamAuthenticationThrottling	Sum	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
IsPrimary	IsPrimary	Histogram	ClusterName , NodeName
JsonBasedCmds	JsonBasedCmds	Sum	ClusterName , NodeName
JsonBased GetCmds	JsonBased GetCmds	Sum	ClusterName , NodeName
JsonBased SetCmds	JsonBased SetCmds	Sum	ClusterName , NodeName
KeyAuthor izationFa ilures	KeyAuthor izationFa ilures	Sum	ClusterName , NodeName
KeyBasedCmds	KeyBasedCmds	Sum	ClusterName , NodeName
KeysTracked	KeysTracked	Histogram	ClusterName , NodeName
KeyspaceHits	KeyspaceHits	Sum	ClusterName , NodeName
KeyspaceMisses	KeyspaceMisses	Sum	ClusterName , NodeName
ListBasedCmds	ListBasedCmds	Sum	ClusterName , NodeName
MaxReplic ationThro ughput	MaxReplic ationThro ughput	Histogram	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MemoryFragmentationRatio	MemoryFragmentationRatio	Histogram	ClusterName , NodeName
NetworkBandwidthInAllowanceExceeded	NetworkBandwidthInAllowanceExceeded	Sum	ClusterName , NodeName
NetworkBandwidthOutAllowanceExceeded	NetworkBandwidthOutAllowanceExceeded	Sum	ClusterName , NodeName
NetworkBytesIn	NetworkBytesIn	Sum	ClusterName , NodeName
NetworkBytesOut	NetworkBytesOut	Sum	ClusterName , NodeName
NetworkConntrackAllowanceExceeded	NetworkConntrackAllowanceExceeded	Sum	ClusterName , NodeName
NetworkMaxBytesIn	NetworkMaxBytesIn	Histogram	ClusterName , NodeName
NetworkMaxBytesOut	NetworkMaxBytesOut	Histogram	ClusterName , NodeName
NetworkMaxPacketsIn	NetworkMaxPacketsIn	Histogram	ClusterName , NodeName
NetworkMaxPacketsOut	NetworkMaxPacketsOut	Histogram	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NetworkPackets In	NetworkPackets In	Sum	ClusterName , NodeName
NetworkPackets Out	NetworkPackets Out	Sum	ClusterName , NodeName
NetworkPackets PerSecondAllowanceExceeded	NetworkPackets PerSecondAllowanceExceeded	Sum	ClusterName , NodeName
NewConnections	NewConnections	Sum	ClusterName , NodeName
NonKeyTypeCmds	NonKeyTypeCmds	Sum	ClusterName , NodeName
NumItemsReadFromDisk	NumItemsReadFromDisk	Sum	ClusterName , NodeName
NumItemsWrittenToDisk	NumItemsWrittenToDisk	Sum	ClusterName , NodeName
PrimaryLinkHealthStatus	PrimaryLinkHealthStatus	Histogram	ClusterName , NodeName
PubSubBasedCmds	PubSubBasedCmds	Sum	ClusterName , NodeName
Reclaimed	Reclaimed	Sum	ClusterName , NodeName
ReplicationBytes	ReplicationBytes	Sum	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReplicationDelayedWriteCommands	ReplicationDelayedWriteCommands	Sum	ClusterName , NodeName
ReplicationLag	ReplicationLag	Histogram	ClusterName , NodeName
SearchBasedCmds	SearchBasedCmds	Sum	ClusterName , NodeName
SearchBasedGetCmds	SearchBasedGetCmds	Sum	ClusterName , NodeName
SearchBasedSetCmds	SearchBasedSetCmds	Sum	ClusterName , NodeName
SearchNumberOfIndexedKeys	SearchNumberOfIndexedKeys	Histogram	ClusterName , NodeName
SearchNumberOfIndexes	SearchNumberOfIndexes	Histogram	ClusterName , NodeName
SearchTotalIndexSize	SearchTotalIndexSize	Histogram	ClusterName , NodeName
SetBasedCmds	SetBasedCmds	Sum	ClusterName , NodeName
SetTypeCmds	SetTypeCmds	Sum	ClusterName , NodeName
SortedSetBasedCmds	SortedSetBasedCmds	Sum	ClusterName , NodeName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StreamBasedCmds	StreamBasedCmds	Sum	ClusterName , NodeName
StringBasedCmds	StringBasedCmds	Sum	ClusterName , NodeName
SuccessfulReadRequestLatency	SuccessfulReadRequestLatency	Histogram	ClusterName , NodeName
SuccessfulWriteRequestLatency	SuccessfulWriteRequestLatency	Histogram	ClusterName , NodeName
SwapUsage	SwapUsage	Histogram	ClusterName , NodeName

AWS/NATGateway

The AWS/NATGateway namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/ec2. All enriched instruments use Delta temporality.

- AWS::EC2::NatGateway

AWS::EC2::NatGateway

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveConnectionCount	ActiveConnectionCount	Histogram	NatGatewayId
BytesInFromDestination	BytesInFromDestination	Sum	NatGatewayId
BytesInFromSource	BytesInFromSource	Sum	NatGatewayId
BytesOutToDestination	BytesOutToDestination	Sum	NatGatewayId
BytesOutToSource	BytesOutToSource	Sum	NatGatewayId
ConnectionAttemptCount	ConnectionAttemptCount	Sum	NatGatewayId
ConnectionEstablishedCount	ConnectionEstablishedCount	Sum	NatGatewayId
ErrorPortAllocation	ErrorPortAllocation	Sum	NatGatewayId
IdleTimeoutCount	IdleTimeoutCount	Sum	NatGatewayId
PacketsDropCount	PacketsDropCount	Sum	NatGatewayId
PacketsInFromDestination	PacketsInFromDestination	Sum	NatGatewayId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PacketsIn FromSource	PacketsIn FromSource	Sum	NatGatewayId
PacketsOu tToDestination	PacketsOu tToDestination	Sum	NatGatewayId
PacketsOu tToSource	PacketsOu tToSource	Sum	NatGatewayId
PeakBytes PerSecond	PeakBytes PerSecond	Histogram	NatGatewayId
PeakPacke tsPerSecond	PeakPacke tsPerSecond	Histogram	NatGatewayId

AWS/NetworkELB

The AWS/NetworkELB namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/elasticloadbalancing. All enriched instruments use Delta temporality.

- AWS::ElasticLoadBalancingV2::LoadBalancer

AWS::ElasticLoadBalancingV2::LoadBalancer

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveFlowCount	ActiveFlowCount	Histogram	Availabil ityZone ,

Metric	OpenTelemetry metric	Instrument type	Data point attributes
			LoadBalancer , TargetGroup
ActiveFlowCount_TCP	ActiveFlowCount_TCP	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
ActiveFlowCount_TLS	ActiveFlowCount_TLS	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
ActiveFlowCount_UDP	ActiveFlowCount_UDP	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
ActiveZonalShiftHostCount	ActiveZonalShiftHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
HealthyHostCount	HealthyHostCount	Histogram	AvailabilityZone , LoadBalancer , TargetGroup
NewFlowCount	NewFlowCount	Sum	AvailabilityZone , LoadBalancer , TargetGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NewFlowCount_TCP	NewFlowCount_TCP	Sum	AvailabilityZone, LoadBalancer, TargetGroup
NewFlowCount_TLS	NewFlowCount_TLS	Sum	AvailabilityZone, LoadBalancer, TargetGroup
NewFlowCount_UDP	NewFlowCount_UDP	Sum	AvailabilityZone, LoadBalancer, TargetGroup
PeakPacketsPerSecond	PeakPacketsPerSecond	Histogram	AvailabilityZone, LoadBalancer
PortAllocationErrorCount	PortAllocationErrorCount	Sum	AvailabilityZone, LoadBalancer
ProcessedBytes	ProcessedBytes	Sum	AvailabilityZone, LoadBalancer
ProcessedBytes_TCP	ProcessedBytes_TCP	Sum	AvailabilityZone, LoadBalancer
ProcessedBytes_TLS	ProcessedBytes_TLS	Sum	AvailabilityZone, LoadBalancer

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ProcessedBytes_UDP	ProcessedBytes_UDP	Sum	AvailabilityZone, LoadBalancer
ProcessedPackets	ProcessedPackets	Sum	AvailabilityZone, LoadBalancer
TCP_Client_Reset_Count	TCP_Client_Reset_Count	Sum	AvailabilityZone, LoadBalancer
TCP_ELB_Reset_Count	TCP_ELB_Reset_Count	Sum	AvailabilityZone, LoadBalancer
TCP_Target_Reset_Count	TCP_Target_Reset_Count	Sum	AvailabilityZone, LoadBalancer
UnHealthyHostCount	UnHealthyHostCount	Histogram	AvailabilityZone, LoadBalancer, TargetGroup

AWS/NetworkFirewall

The AWS/NetworkFirewall namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/network-firewall. All enriched instruments use Delta temporality.

- AWS::NetworkFirewall::Firewall

AWS::NetworkFirewall::Firewall

- cloud.resource_id enrichment: Supported

- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DroppedPackets	DroppedPackets	Sum	AvailabilityZone, Engine, FirewallName
InvalidDroppedPackets	InvalidDroppedPackets	Sum	AvailabilityZone, Engine, FirewallName
OtherDroppedPackets	OtherDroppedPackets	Sum	AvailabilityZone, Engine, FirewallName
Packets	Packets	Sum	AvailabilityZone, CustomAction, Engine, FirewallName
PassedPackets	PassedPackets	Sum	AvailabilityZone, Engine, FirewallName
ReceivedPackets	ReceivedPackets	Sum	AvailabilityZone, Engine, FirewallName
RejectedPackets	RejectedPackets	Sum	AvailabilityZone, Engine, FirewallName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StreamExceptionPolicyPackets	StreamExceptionPolicyPackets	Sum	AvailabilityZone ,Engine, FirewallName
TLSDroppedPackets	TLSDroppedPackets	Sum	AvailabilityZone ,Engine, FirewallName
TLSErrors	TLSErrors	Sum	AvailabilityZone ,Engine, FirewallName
TLSPassedPackets	TLSPassedPackets	Sum	AvailabilityZone ,Engine, FirewallName
TLSReceivedPackets	TLSReceivedPackets	Sum	AvailabilityZone ,Engine, FirewallName
TLSRejectedPackets	TLSRejectedPackets	Sum	AvailabilityZone ,Engine, FirewallName
TLSRevocationStatusOKConnections	TLSRevocationStatusOKConnections	Sum	AvailabilityZone ,Engine, FirewallName
TLSRevocationStatusRevokedConnections	TLSRevocationStatusRevokedConnections	Sum	AvailabilityZone ,Engine, FirewallName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TLSRevocationStatusUnknownConnections	TLSRevocationStatusUnknownConnections	Sum	AvailabilityZone ,Engine, FirewallName
TLSTimedOutConnections	TLSTimedOutConnections	Sum	AvailabilityZone ,Engine, FirewallName

AWS/Pinpoint

The AWS/Pinpoint namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/mobiletargeting. All enriched instruments use Delta temporality.

- AWS::Pinpoint::App

AWS::Pinpoint::App

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CampaignSendMessageLatency	CampaignSendMessageLatency	Histogram	ApplicationId , Channel
CampaignSendMessagePermanentFailure	CampaignSendMessagePermanentFailure	Sum	ApplicationId , Channel

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CampaignSendMessageSuccess	CampaignSendMessageSuccess	Sum	ApplicationId , Channel
CampaignSendMessageTemporaryFailure	CampaignSendMessageTemporaryFailure	Sum	ApplicationId , Channel
CampaignSendMessageThrottled	CampaignSendMessageThrottled	Sum	ApplicationId , Channel
DLRFailure	DLRFailure	Sum	ApplicationId , IsoCountryCode
DLRSuccess	DLRSuccess	Sum	ApplicationId , IsoCountryCode
DestinationChannelCount	DestinationChannelCount	Histogram	ApplicationId , Channel
DirectSendMessageLatency	DirectSendMessageLatency	Histogram	ApplicationId , Channel
DirectSendMessagePermanentFailure	DirectSendMessagePermanentFailure	Sum	ApplicationId , Channel
DirectSendMessageSuccess	DirectSendMessageSuccess	Sum	ApplicationId , Channel

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DirectSendMessageTemporaryFailure	DirectSendMessageTemporaryFailure	Sum	ApplicationId , Channel
DirectSendMessageThrottled	DirectSendMessageThrottled	Sum	ApplicationId , Channel
EndpointRegistrationFailure	EndpointRegistrationFailure	Sum	ApplicationId
ExportEventErrors	ExportEventErrors	Sum	ApplicationId , ErrorCode
ExportedEvents	ExportedEvents	Sum	ApplicationId
ImportJobDuration	ImportJobDuration	Histogram	ApplicationId
ImportJobFailure	ImportJobFailure	Sum	ApplicationId
ImportedEndpointFailure	ImportedEndpointFailure	Sum	ApplicationId
JourneySendMessageLatency	JourneySendMessageLatency	Histogram	ApplicationId , Channel
JourneySendMessagePermanentFailure	JourneySendMessagePermanentFailure	Sum	ApplicationId , Channel

Metric	OpenTelemetry metric	Instrument type	Data point attributes
JourneySendMessageSuccess	JourneySendMessageSuccess	Sum	ApplicationId , Channel
OTPSendMessageLatency	OTPSendMessageLatency	Histogram	ApplicationId , Channel
OTPSendMessageSuccess	OTPSendMessageSuccess	Sum	ApplicationId , Channel
OTPVerificationAttempt	OTPVerificationAttempt	Sum	ApplicationId , Channel
OTPVerificationFailure	OTPVerificationFailure	Sum	ApplicationId , Channel
OTPVerificationFailureFinalAttempt	OTPVerificationFailureFinalAttempt	Sum	ApplicationId
OTPVerificationSuccess	OTPVerificationSuccess	Sum	ApplicationId , Channel
TotalEvents	TotalEvents	Sum	ApplicationId

AWS/Prometheus

The AWS/Prometheus namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/aps. All enriched instruments use Delta temporality.

- AWS::APS::AnomalyDetector
- AWS::APS::RuleGroupsNamespace
- AWS::APS::Workspace

AWS::APS::AnomalyDetector

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`AnomalyDetectorEvaluationFailures`	`AnomalyDetectorEvaluationFailures`	Sum	`AnomalyDetectorId`, `Workspace`
`AnomalyDetectorEvaluations`	`AnomalyDetectorEvaluations`	Sum	`AnomalyDetectorId`, `Workspace`
`AnomalyDetectorLastEvaluationDuration`	`AnomalyDetectorLastEvaluationDuration`	Histogram	`AnomalyDetectorId`, `Workspace`

AWS::APS::RuleGroupsNamespace

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`RuleEvaluationFailures`	`RuleEvaluationFailures`	Sum	`RuleGroup`, `Workspace`
`RuleEvaluations`	`RuleEvaluations`	Sum	`RuleGroup`, `Workspace`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
RuleGroupIterationsMissed	RuleGroupIterationsMissed	Sum	RuleGroup, Workspace
RuleGroupLastEvaluationDuration	RuleGroupLastEvaluationDuration	Histogram	RuleGroup, Workspace

AWS::APS::Workspace

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveSeriesLimitPerLabelSet	ActiveSeriesLimitPerLabelSet	Histogram	LabelSet, Workspace
ActiveSeriesPerLabelSet	ActiveSeriesPerLabelSet	Histogram	LabelSet, Workspace
AlertManagerAlertsReceived	AlertManagerAlertsReceived	Sum	Workspace
AlertManagerNotificationsFailed	AlertManagerNotificationsFailed	Sum	Workspace
AlertManagerNotifi	AlertManagerNotifi	Sum	Workspace

Metric	OpenTelemetry metric	Instrument type	Data point attributes
cationsFailedByIntegration	cationsFailedByIntegration		
AlertManagerNotificationsThrottled	AlertManagerNotificationsThrottled	Sum	Workspace
AlertManagerNotificationsThrottledByIntegration	AlertManagerNotificationsThrottledByIntegration	Sum	Workspace
DiscardedSamples	DiscardedSamples	Sum	Reason, Workspace
DiscardedSamplesPerLabelSet	DiscardedSamplesPerLabelSet	Sum	LabelSet, Reason, Workspace
DiscardedSeries	DiscardedSeries	Sum	Reason, Workspace
DiscardedSeriesPerLabelSet	DiscardedSeriesPerLabelSet	Sum	LabelSet, Reason, Workspace
IngestionRatePerLabelSet	IngestionRatePerLabelSet	Histogram	LabelSet, Workspace
QuerySamplesProcessed	QuerySamplesProcessed	Sum	Workspace

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SecretFetchFailure	SecretFetchFailure	Sum	Workspace

AWS/RDS

The AWS/RDS namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/rds. All enriched instruments use Delta temporality.

- AWS::RDS::DBCluster
- AWS::RDS::DBInstance

AWS::RDS::DBCluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AbortedClients	AbortedClients	Sum	DBClusterIdentifier, Role
ActiveTransactions	ActiveTransactions	Histogram	DBClusterIdentifier, Role
AuroraBinlogReplicaLag	AuroraBinlogReplicaLag	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AuroraDMLRejectedMasterFull	AuroraDMLRejectedMasterFull	Sum	DBClusterIdentifier, Role
AuroraReplicaLagMaximum	AuroraReplicaLagMaximum	Histogram	DBClusterIdentifier, Role
AuroraReplicaLagMinimum	AuroraReplicaLagMinimum	Histogram	DBClusterIdentifier, Role
AuroraVolumeBytesLeftTotal	AuroraVolumeBytesLeftTotal	Histogram	DBClusterIdentifier, Role
Aurora_pq_request_attempted	Aurora_pq_request_attempted	Sum	DBClusterIdentifier, Role
Aurora_pq_request_executed	Aurora_pq_request_executed	Sum	DBClusterIdentifier, Role
Aurora_pq_request_failed	Aurora_pq_request_failed	Sum	DBClusterIdentifier, Role
Aurora_pq_request_in_progress	Aurora_pq_request_in_progress	Histogram	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen	Aurora_pq_request_not_chosen	Sum	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Aurora_pq_request_not_chosen_below_min_rows	Aurora_pq_request_not_chosen_below_min_rows	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_column_bit	Aurora_pq_request_not_chosen_column_bit	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_column_geometry	Aurora_pq_request_not_chosen_column_geometry	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_column_lob	Aurora_pq_request_not_chosen_column_lob	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_column_virtual	Aurora_pq_request_not_chosen_column_virtual	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_custom_charset	Aurora_pq_request_not_chosen_custom_charset	Sum	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Aurora_pq_request_not_chosen_fast_ddl	Aurora_pq_request_not_chosen_fast_ddl	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_full_text_index	Aurora_pq_request_not_chosen_full_text_index	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_high_buffer_pool_pct	Aurora_pq_request_not_chosen_high_buffer_pool_pct	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_index_hint	Aurora_pq_request_not_chosen_index_hint	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_innodb_table_format	Aurora_pq_request_not_chosen_innodb_table_format	Sum	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Aurora_pq_request_not_chosen_long_trx	Aurora_pq_request_not_chosen_long_trx	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_no_where_clause	Aurora_pq_request_not_chosen_no_where_clause	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_range_scan	Aurora_pq_request_not_chosen_range_scan	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_row_length_too_long	Aurora_pq_request_not_chosen_row_length_too_long	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_small_table	Aurora_pq_request_not_chosen_small_table	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_temporary_table	Aurora_pq_request_not_chosen_temporary_table	Sum	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Aurora_pq_request_not_chosen_tx_isolation	Aurora_pq_request_not_chosen_tx_isolation	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_unsupported_access	Aurora_pq_request_not_chosen_unsupported_access	Sum	DBClusterIdentifier, Role
Aurora_pq_request_not_chosen_update_delete_stmts	Aurora_pq_request_not_chosen_update_delete_stmts	Sum	DBClusterIdentifier, Role
Aurora_pq_request_throttled	Aurora_pq_request_throttled	Sum	DBClusterIdentifier, Role
BlockedTransactions	BlockedTransactions	Histogram	DBClusterIdentifier, Role
BufferCacheHitRatio	BufferCacheHitRatio	Histogram	DBClusterIdentifier, Role
CPUUtilization	CPUUtilization	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CommitLatency	CommitLatency	Histogram	DBCluster Identifier , Role
CommitThroughput	CommitThroughput	Histogram	DBCluster Identifier , Role
ConnectionAttempts	ConnectionAttempts	Sum	DBCluster Identifier , Role
DDLLatency	DDLLatency	Histogram	DBCluster Identifier , Role
DDLThroughput	DDLThroughput	Histogram	DBCluster Identifier , Role
DMLLatency	DMLLatency	Histogram	DBCluster Identifier , Role
DMLThroughput	DMLThroughput	Histogram	DBCluster Identifier , Role
DatabaseConnections	DatabaseConnections	Histogram	DBCluster Identifier , Role
Deadlocks	Deadlocks	Histogram	DBCluster Identifier , Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeleteLatency	DeleteLatency	Histogram	DBCluster Identifier , Role
DeleteThroughput	DeleteThroughput	Histogram	DBCluster Identifier , Role
DiskQueueDepth	DiskQueueDepth	Histogram	DBCluster Identifier , Role
EBSByteBalance%	EBSByteBalance%	Histogram	DBCluster Identifier , Role
EBSIOBalance%	EBSIOBalance%	Histogram	DBCluster Identifier , Role
EngineUptime	EngineUptime	Histogram	DBCluster Identifier , Role
ForwardingMasterDMLLatency	ForwardingMasterDMLLatency	Histogram	DBCluster Identifier , Role
ForwardingMasterDMLThroughput	ForwardingMasterDMLThroughput	Histogram	DBCluster Identifier , Role
ForwardingMasterOpenSessions	ForwardingMasterOpenSessions	Histogram	DBCluster Identifier , Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ForwardingReplicaDMLLatency	ForwardingReplicaDMLLatency	Histogram	DBClusterIdentifier, Role
ForwardingReplicaDMLThroughput	ForwardingReplicaDMLThroughput	Histogram	DBClusterIdentifier, Role
ForwardingReplicaOpenSessions	ForwardingReplicaOpenSessions	Histogram	DBClusterIdentifier, Role
ForwardingReplicaReadWaitLatency	ForwardingReplicaReadWaitLatency	Histogram	DBClusterIdentifier, Role
ForwardingReplicaReadWaitThroughput	ForwardingReplicaReadWaitThroughput	Histogram	DBClusterIdentifier, Role
ForwardingReplicaSelectLatency	ForwardingReplicaSelectLatency	Histogram	DBClusterIdentifier, Role
ForwardingReplicaSelectThroughput	ForwardingReplicaSelectThroughput	Histogram	DBClusterIdentifier, Role
FreeLocalStorage	FreeLocalStorage	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FreeableMemory	FreeableMemory	Histogram	DBCluster Identifier , Role
InsertLatency	InsertLatency	Histogram	DBCluster Identifier , Role
InsertThroughput	InsertThroughput	Histogram	DBCluster Identifier , Role
LoginFailures	LoginFailures	Histogram	DBCluster Identifier , Role
NetworkReceiveThroughput	NetworkReceiveThroughput	Histogram	DBCluster Identifier , Role
NetworkThroughput	NetworkThroughput	Histogram	DBCluster Identifier , Role
NetworkTransmitThroughput	NetworkTransmitThroughput	Histogram	DBCluster Identifier , Role
NumBinaryLogFiles	NumBinaryLogFiles	Histogram	DBCluster Identifier , Role
PurgeBoundary	PurgeBoundary	Histogram	DBCluster Identifier , Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PurgeFinishedPoint	PurgeFinishedPoint	Histogram	DBClusterIdentifier, Role
Queries	Queries	Histogram	DBClusterIdentifier, Role
ReadIOPS	ReadIOPS	Histogram	DBClusterIdentifier, Role
ReadLatency	ReadLatency	Histogram	DBClusterIdentifier, Role
ReadThroughput	ReadThroughput	Histogram	DBClusterIdentifier, Role
ResultSetCacheHitRatio	ResultSetCacheHitRatio	Histogram	DBClusterIdentifier, Role
RollbackSegmentHistoryListLength	RollbackSegmentHistoryListLength	Histogram	DBClusterIdentifier, Role
RowLockTime	RowLockTime	Sum	DBClusterIdentifier, Role
SelectLatency	SelectLatency	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
SelectThroughput	SelectThroughput	Histogram	DBClusterIdentifier, Role
StorageNetworkReceiveThroughput	StorageNetworkReceiveThroughput	Histogram	DBClusterIdentifier, Role
StorageNetworkThroughput	StorageNetworkThroughput	Histogram	DBClusterIdentifier, Role
StorageNetworkTransmitThroughput	StorageNetworkTransmitThroughput	Histogram	DBClusterIdentifier, Role
SumBinaryLogSize	SumBinaryLogSize	Histogram	DBClusterIdentifier, Role
SwapUsage	SwapUsage	Histogram	DBClusterIdentifier, Role
TruncateFinishedPoint	TruncateFinishedPoint	Histogram	DBClusterIdentifier, Role
UpdateLatency	UpdateLatency	Histogram	DBClusterIdentifier, Role
UpdateThroughput	UpdateThroughput	Histogram	DBClusterIdentifier, Role

Metric	OpenTelemetry metric	Instrument type	Data point attributes
VolumeBytesUsed	VolumeBytesUsed	Histogram	DbCluster Identifier , EngineName
VolumeReadIOPs	VolumeReadIOPs	Histogram	DbCluster Identifier , EngineName
VolumeWriteIOPs	VolumeWriteIOPs	Histogram	DbCluster Identifier , EngineName
WriteIOPS	WriteIOPS	Histogram	DBCluster Identifier , Role
WriteLatency	WriteLatency	Histogram	DBCluster Identifier , Role
WriteThroughput	WriteThroughput	Histogram	DBCluster Identifier , Role

AWS::RDS::DBInstance

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ACUUtilization	ACUUtilization	Histogram	DBInstanceIdentifier
AuroraEstimatedSharedMemoryBytes	AuroraEstimatedSharedMemoryBytes	Histogram	DBInstanceIdentifier
AuroraReplicaLag	AuroraReplicaLag	Histogram	DBInstanceIdentifier
BurstBalance	BurstBalance	Histogram	DBInstanceIdentifier
CPUUtilization	CPUUtilization	Histogram	DBInstanceIdentifier
DBLoad	DBLoad	Histogram	DBInstanceIdentifier
DBLoadCPU	DBLoadCPU	Histogram	DBInstanceIdentifier
DBLoadNonCPU	DBLoadNonCPU	Histogram	DBInstanceIdentifier
DBLoadRelativeToNumVCPUs	DBLoadRelativeToNumVCPUs	Histogram	DBInstanceIdentifier
DatabaseConnections	DatabaseConnections	Histogram	DBInstanceIdentifier
FreeStorageSpace	FreeStorageSpace	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FreeableMemory	FreeableMemory	Histogram	DBInstanceIdentifier
MaximumUsedTransactionIDs	MaximumUsedTransactionIDs	Histogram	DBInstanceIdentifier
OldestReplicationSlotLag	OldestReplicationSlotLag	Histogram	DBInstanceIdentifier
ReadIOPS	ReadIOPS	Histogram	DBInstanceIdentifier
ReadLatency	ReadLatency	Histogram	DBInstanceIdentifier
ReadThroughput	ReadThroughput	Histogram	DBInstanceIdentifier
ReplicaLag	ReplicaLag	Histogram	DBInstanceIdentifier
TransactionLogsDiskUsage	TransactionLogsDiskUsage	Histogram	DBInstanceIdentifier
TransactionLogsGeneration	TransactionLogsGeneration	Histogram	DBInstanceIdentifier
WriteIOPS	WriteIOPS	Histogram	DBInstanceIdentifier
WriteLatency	WriteLatency	Histogram	DBInstanceIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WriteThroughput	WriteThroughput	Histogram	DBInstanceIdentifier

AWS/RUM

The AWS/RUM namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/rum. All enriched instruments use Delta temporality.

- AWS::RUM::AppMonitor

AWS::RUM::AppMonitor

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Http4xxCount	Http4xxCount	Sum	application_name
Http5xxCount	Http5xxCount	Sum	application_name
HttpStatusCodeCount	HttpStatusCodeCount	Sum	application_name , event_details.response.status , event_type
JsErrorCount	JsErrorCount	Sum	application_name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NavigationFrustratedTransaction	NavigationFrustratedTransaction	Sum	application_name
NavigationSatisfiedTransaction	NavigationSatisfiedTransaction	Sum	application_name
NavigationToleratedTransaction	NavigationToleratedTransaction	Sum	application_name
PageViewCount	PageViewCount	Sum	application_name
PerformanceNavigationDuration	PerformanceNavigationDuration	Histogram	application_name
PerformanceResourceDuration	PerformanceResourceDuration	Histogram	application_name , event_details.file.type , event_type
RumEventPayloadSize	RumEventPayloadSize	Histogram	application_name
SessionCount	SessionCount	Sum	application_name
WebVitalsCumulativeLayoutShift	WebVitalsCumulativeLayoutShift	Histogram	application_name

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WebVitals FirstInputDelay	WebVitals FirstInputDelay	Histogram	application_name
WebVitals LargestContentfulPaint	WebVitals LargestContentfulPaint	Histogram	application_name

AWS/Redshift

The AWS/Redshift namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/redshift. All enriched instruments use Delta temporality.

- AWS::Redshift::Cluster

AWS::Redshift::Cluster

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
CPUUtilization	CPUUtilization	Histogram	ClusterIdentifier ,NodeID
CommitQueueLength	CommitQueueLength	Histogram	ClusterIdentifier
ConcurrencyScalingActiveClusters	ConcurrencyScalingActiveClusters	Histogram	ClusterIdentifier

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConcurrencyScalingSeconds	ConcurrencyScalingSeconds	Sum	ClusterIdentifier
DatabaseConnections	DatabaseConnections	Histogram	ClusterIdentifier ,NodeID
HealthStatus	HealthStatus	Histogram	ClusterIdentifier ,NodeID
MaintenanceMode	MaintenanceMode	Histogram	ClusterIdentifier ,NodeID
MaxConfiguredConcurrencyScalingClusters	MaxConfiguredConcurrencyScalingClusters	Histogram	ClusterIdentifier
NetworkReceiveThroughput	NetworkReceiveThroughput	Histogram	ClusterIdentifier ,NodeID
NetworkTransmitThroughput	NetworkTransmitThroughput	Histogram	ClusterIdentifier ,NodeID
NumExceededSchemaQuotas	NumExceededSchemaQuotas	Histogram	ClusterIdentifier
PercentageDiskSpaceUsed	PercentageDiskSpaceUsed	Histogram	ClusterIdentifier ,NodeID
QueriesCompletedPerSecond	QueriesCompletedPerSecond	Histogram	ClusterIdentifier ,NodeID, latency

Metric	OpenTelemetry metric	Instrument type	Data point attributes
QueryDuration	QueryDuration	Histogram	ClusterId entifier ,NodeID, latency
QueryRuntimeBreakdown	QueryRuntimeBreakdown	Histogram	ClusterId entifier ,NodeID, stage
ReadIOPS	ReadIOPS	Histogram	ClusterId entifier ,NodeID
ReadLatency	ReadLatency	Histogram	ClusterId entifier ,NodeID
ReadThroughput	ReadThroughput	Histogram	ClusterId entifier ,NodeID
RedshiftManagedStorageTotalCapacity	RedshiftManagedStorageTotalCapacity	Histogram	ClusterId entifier
TotalTableCount	TotalTableCount	Histogram	ClusterId entifier
WLMQuerisCompletedPerSecond	WLMQuerisCompletedPerSecond	Histogram	ClusterId entifier ,wlmid
WLMQueryDuration	WLMQueryDuration	Histogram	ClusterId entifier ,NodeID, wlmid
WLMQueueLength	WLMQueueLength	Histogram	ClusterId entifier , QueueName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
WLMQueueWaitTime	WLMQueueWaitTime	Histogram	ClusterIdentifier ,wlmid
WLMRunningQueries	WLMRunningQueries	Histogram	ClusterIdentifier , QueueName
WriteIOPS	WriteIOPS	Histogram	ClusterIdentifier ,NodeID
WriteLatency	WriteLatency	Histogram	ClusterIdentifier ,NodeID
WriteThroughput	WriteThroughput	Histogram	ClusterIdentifier ,NodeID

AWS/Redshift-Serverless

The AWS/Redshift-Serverless namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/redshift-serverless. All enriched instruments use Delta temporality.

- AWS::RedshiftServerless::Namespace
- AWS::RedshiftServerless::Workgroup

AWS::RedshiftServerless::Namespace

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DataStorage	DataStorage	Histogram	Namespace
TotalTableCount	TotalTableCount	Histogram	DatabaseName , Namespace

AWS::RedshiftServerless::Workgroup

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ComputeCapacity	ComputeCapacity	Histogram	Workgroup
ComputeSeconds	ComputeSeconds	Sum	Workgroup
DatabaseConnections	DatabaseConnections	Histogram	DatabaseName , Workgroup
QueriesCompletedPerSecond	QueriesCompletedPerSecond	Histogram	DatabaseName , LatencyRange , Workgroup
QueriesFailed	QueriesFailed	Sum	DatabaseName , QueryType , Workgroup
QueriesQueued	QueriesQueued	Histogram	DatabaseName , QueryType , Workgroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
QueriesRunning	QueriesRunning	Histogram	DatabaseName , QueryType , Workgroup
QueriesSucceeded	QueriesSucceeded	Sum	DatabaseName , QueryType , Workgroup
QueryDuration	QueryDuration	Histogram	DatabaseName , LatencyRange , Workgroup
QueryRuntimeBreakdown	QueryRuntimeBreakdown	Sum	DatabaseName , Workgroup ,stage
UsageLimitAvailable	UsageLimitAvailable	Histogram	UsageLimitId , UsageType , Workgroup
UsageLimitConsumed	UsageLimitConsumed	Histogram	UsageLimitId , UsageType , Workgroup

AWS/Route53

The AWS/Route53 namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/route53. All enriched instruments use Delta temporality.

- AWS::Route53::HealthCheck

AWS::Route53::HealthCheck

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ChildHealthCheckHealthyCount	ChildHealthCheckHealthyCount	Histogram	HealthCheckId
ConnectionTime	ConnectionTime	Histogram	HealthCheckId , Region
HealthCheckPercentageHealthy	HealthCheckPercentageHealthy	Histogram	HealthCheckId
HealthCheckStatus	HealthCheckStatus	Histogram	HealthCheckId
SSLHandshakeTime	SSLHandshakeTime	Histogram	HealthCheckId , Region
TimeToFirstByte	TimeToFirstByte	Histogram	HealthCheckId , Region

AWS/Route53Resolver

The AWS/Route53Resolver namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/route53resolver. All enriched instruments use Delta temporality.

- AWS::Route53Resolver::FirewallRuleGroup
- AWS::Route53Resolver::ResolverEndpoint

AWS::Route53Resolver::FirewallRuleGroup

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FirewallRuleGroupQueryVolume	FirewallRuleGroupQueryVolume	Sum	FirewallRuleGroupId
FirewallRuleGroupVpcQueryVolume	FirewallRuleGroupVpcQueryVolume	Sum	FirewallRuleGroupId , VpcId

AWS::Route53Resolver::ResolverEndpoint

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
AverageResponseTime	AverageResponseTime	Histogram	EndpointId , RniId
CapacityUsage_BETA	CapacityUsage_BETA	Histogram	EndpointId , RniId
ConntrackUtilization	ConntrackUtilization	Histogram	EndpointId , RniId
EndpointHealthyENICount	EndpointHealthyENICount	Histogram	EndpointId
EndpointUnhealthyENICount	EndpointUnhealthyENICount	Histogram	EndpointId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FormErrCount	FormErrCount	Sum	EndpointId , RniId
FormError Queries	FormError Queries	Sum	EndpointId , RniId
InboundQu eryVolume	InboundQu eryVolume	Sum	EndpointId
NXDomainCount	NXDomainCount	Sum	EndpointId , RniId
NoErrorCount	NoErrorCount	Sum	EndpointId , RniId
NxDomainQueries	NxDomainQueries	Sum	EndpointId , RniId
OutboundQ ueryAggre gateVolume	OutboundQ ueryAggre gateVolume	Sum	EndpointId
OutboundQ ueryVolume	OutboundQ ueryVolume	Sum	EndpointId
P90ResponseTime	P90ResponseTime	Histogram	• EndpointI D ,TargetNam eServerIP • EndpointId , RniId
RefusedCount	RefusedCount	Sum	EndpointId , RniId
RefusedQueries	RefusedQueries	Sum	EndpointId , RniId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ReplyCount	ReplyCount	Sum	EndpointId , RniId
RequestQueries	RequestQueries	Sum	EndpointID ,TargetNameServerIP
ResolverEndpointCapacityStatus	ResolverEndpointCapacityStatus	Histogram	EndpointId
ServFailQueries	ServFailQueries	Sum	EndpointId , RniId
ServfailCount	ServfailCount	Sum	EndpointId , RniId
TimeoutCount	TimeoutCount	Sum	EndpointId , RniId
TimeoutQueries	TimeoutQueries	Sum	• EndpointID ,TargetNameServerIP • EndpointId , RniId
tcpRequestCount	tcpRequestCount	Sum	EndpointId , RniId
udpRequestCount	udpRequestCount	Sum	EndpointId , RniId

AWS/S3

The `AWS/S3` namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/s3`. All enriched instruments use Delta temporality.

- AWS::S3::Bucket

AWS::S3::Bucket

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
`4xxErrors`	`4xxErrors`	Sum	`BucketName` , `FilterId`
`5xxErrors`	`5xxErrors`	Sum	`BucketName` , `FilterId`
`AllRequests`	`AllRequests`	Sum	`BucketName` , `FilterId`
`BucketSizeBytes`	`BucketSizeBytes`	Histogram	`BucketName` , `StorageType`
`BytesDownloaded`	`BytesDownloaded`	Sum	`BucketName` , `FilterId`
`BytesPendingReplication`	`BytesPendingReplication`	Histogram	`DestinationBucket` , `RuleId`, `SourceBucket`
`BytesUploaded`	`BytesUploaded`	Sum	`BucketName` , `FilterId`

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeleteRequests	DeleteRequests	Sum	BucketName , FilterId
FirstByte Latency	FirstByte Latency	Histogram	BucketName , FilterId
GetRequests	GetRequests	Sum	BucketName , FilterId
HeadRequests	HeadRequests	Sum	BucketName , FilterId
ListRequests	ListRequests	Sum	BucketName , FilterId
NumberOfObjects	NumberOfObjects	Histogram	BucketName , StorageType
Operation sFailedRe plication	Operation sFailedRe plication	Sum	Destinati onBucket , RuleId, SourceBucket
Operation sPendingR eplication	Operation sPendingR eplication	Histogram	Destinati onBucket , RuleId, SourceBucket
PostRequests	PostRequests	Sum	BucketName , FilterId
PutRequests	PutRequests	Sum	BucketName , FilterId
Replicati onLatency	Replicati onLatency	Histogram	Destinati onBucket , RuleId, SourceBucket

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalRequestLatency	TotalRequestLatency	Histogram	BucketName , FilterId

AWS/SNS

The AWS/SNS namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/sns. All enriched instruments use Delta temporality.

- AWS::SNS::Topic

AWS::SNS::Topic

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NumberOfMessagesPublished	NumberOfMessagesPublished	Sum	TopicName
NumberOfNotificationsDelivered	NumberOfNotificationsDelivered	Sum	TopicName
NumberOfNotificationsFailed	NumberOfNotificationsFailed	Sum	TopicName
PublishSize	PublishSize	Histogram	TopicName

AWS/SQS

The AWS/SQS namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/sqs`. All enriched instruments use Delta temporality.

- AWS::SQS::Queue

AWS::SQS::Queue

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ApproximateAgeOfOldestMessage	ApproximateAgeOfOldestMessage	Histogram	QueueName
ApproximateAgeOfOldestMessageInQuietGroups	ApproximateAgeOfOldestMessageInQuietGroups	Histogram	QueueName
ApproximateNumberOfGroupsWithInflightMessages	ApproximateNumberOfGroupsWithInflightMessages	Histogram	QueueName
ApproximateNumberOfMessagesDelayed	ApproximateNumberOfMessagesDelayed	Histogram	QueueName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ApproximateNumberOfMessagesDelayedInQuietGroups	ApproximateNumberOfMessagesDelayedInQuietGroups	Histogram	QueueName
ApproximateNumberOfMessagesNotVisible	ApproximateNumberOfMessagesNotVisible	Histogram	QueueName
ApproximateNumberOfMessagesNotVisibleInQuietGroups	ApproximateNumberOfMessagesNotVisibleInQuietGroups	Histogram	QueueName
ApproximateNumberOfMessagesVisible	ApproximateNumberOfMessagesVisible	Histogram	QueueName
ApproximateNumberOfMessagesVisibleInQuietGroups	ApproximateNumberOfMessagesVisibleInQuietGroups	Histogram	QueueName
ApproximateNumberOfNoisyGroups	ApproximateNumberOfNoisyGroups	Histogram	QueueName
NumberOfDeduplicatedSentMessages	NumberOfDeduplicatedSentMessages	Sum	QueueName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
NumberOfEmptyReceives	NumberOfEmptyReceives	Sum	QueueName
NumberOfMessagesDeleted	NumberOfMessagesDeleted	Sum	QueueName
NumberOfMessagesReceived	NumberOfMessagesReceived	Sum	QueueName
NumberOfMessagesSent	NumberOfMessagesSent	Sum	QueueName
SentMessageSize	SentMessageSize	Histogram	QueueName

AWS/SWF

The AWS/SWF namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/swf`. All enriched instruments use Delta temporality.

- domain

domain

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActivityTaskScheduleToCloseTime	ActivityTaskScheduleToCloseTime	Histogram	ActivityTypeName , ActivityT

Metric	OpenTelemetry metric	Instrument type	Data point attributes
			ypeVersion , Domain
ActivityTaskScheduleToStartTime	ActivityTaskScheduleToStartTime	Histogram	ActivityTypeName , ActivityTypeVersion , Domain
ActivityTaskStartToCloseTime	ActivityTaskStartToCloseTime	Histogram	ActivityTypeName , ActivityTypeVersion , Domain
ActivityTasksCanceled	ActivityTasksCanceled	Sum	ActivityTypeName , ActivityTypeVersion , Domain
ActivityTasksCompleted	ActivityTasksCompleted	Sum	ActivityTypeName , ActivityTypeVersion , Domain
ActivityTasksFailed	ActivityTasksFailed	Sum	ActivityTypeName , ActivityTypeVersion , Domain

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DecisionTaskScheduleToStartTime	DecisionTaskScheduleToStartTime	Histogram	Domain, WorkflowTypeName , WorkflowTypeVersion
DecisionTaskStartToCloseTime	DecisionTaskStartToCloseTime	Histogram	Domain, WorkflowTypeName , WorkflowTypeVersion
DecisionTasksCompleted	DecisionTasksCompleted	Sum	Domain, WorkflowTypeName , WorkflowTypeVersion
PendingTasks	PendingTasks	Sum	Domain, TaskListName
ScheduledActivityTasksTimedOutOnClose	ScheduledActivityTasksTimedOutOnClose	Sum	ActivityTypeName , ActivityTypeVersion , Domain
ScheduledActivityTasksTimedOutOnStart	ScheduledActivityTasksTimedOutOnStart	Sum	ActivityTypeName , ActivityTypeVersion , Domain
StartedActivityTasksTimedOutOnClose	StartedActivityTasksTimedOutOnClose	Sum	ActivityTypeName , ActivityTypeVersion , Domain

Metric	OpenTelemetry metric	Instrument type	Data point attributes
StartedActivityTasksTimedOutOnHeartbeat	StartedActivityTasksTimedOutOnHeartbeat	Sum	ActivityTypeName, ActivityTypeVersion, Domain
StartedDecisionTasksTimedOutOnClose	StartedDecisionTasksTimedOutOnClose	Sum	Domain, WorkflowTypeName, WorkflowTypeVersion
WorkflowStartToCloseTime	WorkflowStartToCloseTime	Histogram	Domain, WorkflowTypeName, WorkflowTypeVersion
WorkflowsCanceled	WorkflowsCanceled	Sum	Domain, WorkflowTypeName, WorkflowTypeVersion
WorkflowsCompleted	WorkflowsCompleted	Sum	Domain, WorkflowTypeName, WorkflowTypeVersion
WorkflowsContinuedAsNew	WorkflowsContinuedAsNew	Sum	Domain, WorkflowTypeName, WorkflowTypeVersion
WorkflowsFailed	WorkflowsFailed	Sum	Domain, WorkflowTypeName, WorkflowTypeVersion

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Workflows Terminated	Workflows Terminated	Sum	Cause, Domain, WorkflowTypeName , WorkflowTypeVersion
Workflows TimedOut	Workflows TimedOut	Sum	Domain, WorkflowTypeName , WorkflowTypeVersion

AWS/SageMaker

The AWS/SageMaker namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/sagemaker. All enriched instruments use Delta temporality.

- AWS::SageMaker::Endpoint
- AWS::SageMaker::InferenceComponent

AWS::SageMaker::Endpoint

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConcurrentRequestsPerModel	ConcurrentRequestsPerModel	Histogram	EndpointName , VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConcurrentRequestsPerModel	ConcurrentRequestsPerModelByAzByInstanceType	Histogram	AvailabilityZone , EndpointName , InstanceType , Region, VariantName
FirstChunkLatency	FirstChunkLatency	Histogram	EndpointName , VariantName
FirstChunkLatency	FirstChunkLatencyByAzByInstanceType	Histogram	AvailabilityZone , EndpointName , InstanceType , Region, VariantName
FirstChunkModelLatency	FirstChunkModelLatency	Histogram	EndpointName , VariantName
FirstChunkModelLatency	FirstChunkModelLatencyByAzByInstanceType	Histogram	AvailabilityZone , EndpointName , InstanceType , Region, VariantName
FirstChunkOverheadLatency	FirstChunkOverheadLatency	Histogram	EndpointName , VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
FirstChunkOverheadLatency	FirstChunkOverheadLatencyByAzByInstanceType	Histogram	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
Invocation4XXErrors	Invocation4XXErrors	Sum	EndpointName, VariantName
Invocation4XXErrors	Invocation4XXErrorsByAzByInstanceType	Sum	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
Invocation4XXErrors	Invocation4XXErrorsByInstanceId	Sum	EndpointName, InstanceId, VariantName
Invocation4XXErrors	Invocation4XXErrorsForIcByInstanceIdByContainerId	Sum	ContainerId, EndpointName, InferenceComponentName, InstanceId, VariantName
Invocation5XXErrors	Invocation5XXErrors	Sum	EndpointName, VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Invocation5XXErrors	Invocation5XXErrorsByAzByInstanceType	Sum	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
Invocation5XXErrors	Invocation5XXErrorsByInstanceId	Sum	EndpointName, InstanceId, VariantName
Invocation5XXErrors	Invocation5XXErrorsForIcByInstanceIdByContainerId	Sum	ContainerId, EndpointName, InferenceComponentName, InstanceId, VariantName
InvocationModelErrors	InvocationModelErrors	Sum	EndpointName, VariantName
InvocationModelErrors	InvocationModelErrorsByAzByInstanceType	Sum	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
InvocationModelErrors	InvocationModelErrorsByInstanceId	Sum	EndpointName, InstanceId, VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InvocationModelErrors	InvocationModelErrorsForIcByInstanceIdByContainerId	Sum	ContainerId, EndpointName, InferenceComponentName, InstanceId, VariantName
Invocations	Invocations	Sum	EndpointName, VariantName
Invocations	InvocationsByAzByInstanceType	Sum	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
Invocations	InvocationsByInstanceId	Sum	EndpointName, InstanceId, VariantName
Invocations	InvocationsForIcByInstanceIdByContainerId	Sum	ContainerId, EndpointName, InferenceComponentName, InstanceId, VariantName
InvocationsPerInstance	InvocationsPerInstance	Sum	EndpointName, VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InvocationsPerInstance	InvocationsPerInstanceByAzByInstanceType	Sum	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
InvocationsPerInstance	InvocationsPerInstanceByInstanceId	Sum	EndpointName, InstanceId, VariantName
MidStreamErrors	MidStreamErrors	Sum	EndpointName, VariantName
ModelLatency	ModelLatency	Histogram	EndpointName, VariantName
ModelLatency	ModelLatencyByAzByInstanceType	Histogram	AvailabilityZone, EndpointName, InstanceType, Region, VariantName
ModelLatency	ModelLatencyByInstanceId	Histogram	EndpointName, InstanceId, VariantName
ModelLatency	ModelLatencyForIcByInstanceIdByContainerId	Histogram	ContainerId, EndpointName, InferenceComponentName, InstanceId, VariantName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OverheadLatency	OverheadLatency	Histogram	EndpointName , VariantName
OverheadLatency	OverheadLatencyByAzByInstanceType	Histogram	AvailabilityZone , EndpointName , InstanceType , Region, VariantName
OverheadLatency	OverheadLatencyByInstanceId	Histogram	EndpointName , InstanceId , VariantName
OverheadLatency	OverheadLatencyForIcByInstanceIdByContainerId	Histogram	ContainerId ,EndpointName ,InferenceComponentName , InstanceId , VariantName

AWS::SageMaker::InferenceComponent

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConcurrentRequestsPerCopy	ConcurrentRequestsPerCopy	Histogram	InferenceComponentName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConcurrentRequestsPerCopy	ConcurrentRequestsPerCopyByAzByInstanceType	Histogram	AvailabilityZone, InferenceComponentName, InstanceType, Region
Invocation4XXErrors	Invocation4XXErrorsForIc	Sum	InferenceComponentName
Invocation4XXErrors	Invocation4XXErrorsForIcByAzByInstanceType	Sum	AvailabilityZone, InferenceComponentName, InstanceType, Region
Invocation5XXErrors	Invocation5XXErrorsForIc	Sum	InferenceComponentName
Invocation5XXErrors	Invocation5XXErrorsForIcByAzByInstanceType	Sum	AvailabilityZone, InferenceComponentName, InstanceType, Region
InvocationModelErrors	InvocationModelErrorsForIc	Sum	InferenceComponentName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InvocationModelErrors	InvocationModelErrorsForIcByAzByInstanceType	Sum	AvailabilityZone, InferenceComponentName, InstanceType, Region
Invocations	InvocationsForIc	Sum	InferenceComponentName
Invocations	InvocationsForIcByAzByInstanceType	Sum	AvailabilityZone, InferenceComponentName, InstanceType, Region
InvocationsPerCopy	InvocationsPerCopy	Sum	InferenceComponentName
InvocationsPerCopy	InvocationsPerCopyByAzByInstanceType	Sum	AvailabilityZone, InferenceComponentName, InstanceType, Region
ModelLatency	ModelLatencyForIc	Histogram	InferenceComponentName

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ModelLatency	ModelLatencyForIcByAzByInstanceType	Histogram	AvailabilityZone, InferenceComponentName, InstanceType, Region
OverheadLatency	OverheadLatencyForIc	Histogram	InferenceComponentName
OverheadLatency	OverheadLatencyForIcByAzByInstanceType	Histogram	AvailabilityZone, InferenceComponentName, InstanceType, Region

AWS/Scheduler

The AWS/Scheduler namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/scheduler. All enriched instruments use Delta temporality.

- AWS::Scheduler::ScheduleGroup

AWS::Scheduler::ScheduleGroup

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Invocationattemptcount	Invocationattemptcount	Sum	ScheduleGroup
Invocationdroppedcount	Invocationdroppedcount	Sum	ScheduleGroup
Invocationthrottlecount	Invocationthrottlecount	Sum	ScheduleGroup
Invocationsfailedtobesenttodeadlettercount	Invocationsfailedtobesenttodeadlettercount	Sum	ScheduleGroup
Invocationsfailedtobesenttodeadlettercount_AWS.SimpleQueueService.NonExistentQueue	Invocationsfailedtobesenttodeadlettercount_AWS.SimpleQueueService.NonExistentQueue	Sum	ScheduleGroup
Invocationsfailedtobesenttodeadlettercount_AccessDenied	Invocationsfailedtobesenttodeadlettercount_AccessDenied	Sum	ScheduleGroup
Invocationsfailedtobesentto	Invocationsfailedtobesentto	Sum	ScheduleGroup

Metric	OpenTelemetry metric	Instrument type	Data point attributes
DeadLetterCount_KMS.AccessDeniedException	DeadLetterCount_KMS.AccessDeniedException		
InvocationsSentToDeadLetterCount	InvocationsSentToDeadLetterCount	Sum	ScheduleGroup
TargetErrorCount	TargetErrorCount	Sum	ScheduleGroup
TargetErrorThrottledCount	TargetErrorThrottledCount	Sum	ScheduleGroup

AWS/Transfer

The AWS/Transfer namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/transfer. All enriched instruments use Delta temporality.

- AWS::Transfer::Connector
- AWS::Transfer::Server

AWS::Transfer::Connector

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OutboundFailedMessage	OutboundFailedMessage	Sum	ConnectorId
OutboundMessage	OutboundMessage	Sum	ConnectorId

AWS::Transfer::Server

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesIn	BytesIn	Sum	ServerId
BytesOut	BytesOut	Sum	ServerId
ConcurrentSessions	ConcurrentSessions	Histogram	ServerId
FilesIn	FilesIn	Sum	ServerId
FilesOut	FilesOut	Sum	ServerId
InboundFailedMessage	InboundFailedMessage	Sum	ServerId
InboundMessage	InboundMessage	Sum	ServerId
OnUploadExecutionsFailed	OnUploadExecutionsFailed	Sum	ServerId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
OnUploadExecutionsStarted	OnUploadExecutionsStarted	Sum	ServerId
OnUploadExecutionsSuccess	OnUploadExecutionsSuccess	Sum	ServerId
OutboundFailedMessage	OutboundFailedMessage	Sum	ServerId

AWS/TransitGateway

The AWS/TransitGateway namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/ec2. All enriched instruments use Delta temporality.

- AWS::EC2::TransitGateway

AWS::EC2::TransitGateway

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesDropCountBlackhole	BytesDropCountBlackhole	Sum	AvailabilityZone , TransitGateway ,TransitGatewayAttachment

Metric	OpenTelemetry metric	Instrument type	Data point attributes
BytesDropCountInternalError	BytesDropCountInternalError	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
BytesDropCountNoPolicy	BytesDropCountNoPolicy	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
BytesDropCountNoRoute	BytesDropCountNoRoute	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
BytesIn	BytesIn	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
BytesOut	BytesOut	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PacketDropCountBlackhole	PacketDropCountBlackhole	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
PacketDropCountInternalError	PacketDropCountInternalError	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
PacketDropCountNoPolicy	PacketDropCountNoPolicy	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
PacketDropCountNoRoute	PacketDropCountNoRoute	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
PacketDropCountTTLExpired	PacketDropCountTTLExpired	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment

Metric	OpenTelemetry metric	Instrument type	Data point attributes
PacketsIn	PacketsIn	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment
PacketsOut	PacketsOut	Sum	AvailabilityZone, TransitGateway, TransitGatewayAttachment

AWS/VPN

The AWS/VPN namespace is published under the OpenTelemetry instrumentation scope `cloudwatch.aws/ec2`. All enriched instruments use Delta temporality.

- AWS::EC2::VPNConnection

AWS::EC2::VPNConnection

- `cloud.resource_id` enrichment: Supported
- `tag.*` enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TunnelDataIn	TunnelDataIn	Sum	VpnId
TunnelDataOut	TunnelDataOut	Sum	VpnId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TunnelState	TunnelState	Histogram	VpnId

AWS/VpcLattice

The AWS/VpcLattice namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/vpc-lattice. All enriched instruments use Delta temporality.

- AWS::VpcLattice::Service

AWS::VpcLattice::Service

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
HTTPCode_2XX_Count	HTTPCode_2XX_Count	Sum	AvailabilityZone ,Service
HTTPCode_3XX_Count	HTTPCode_3XX_Count	Sum	AvailabilityZone ,Service
HTTPCode_4XX_Count	HTTPCode_4XX_Count	Sum	AvailabilityZone ,Service
HTTPCode_5XX_Count	HTTPCode_5XX_Count	Sum	AvailabilityZone ,Service
RequestTime	RequestTime	Histogram	AvailabilityZone ,Service
RequestTimeoutCount	RequestTimeoutCount	Sum	AvailabilityZone ,Service

Metric	OpenTelemetry metric	Instrument type	Data point attributes
TotalRequestCount	TotalRequestCount	Sum	AvailabilityZone, Service

AWS/WorkSpaces

The AWS/WorkSpaces namespace is published under the OpenTelemetry instrumentation scope cloudwatch.aws/workspaces. All enriched instruments use Delta temporality.

- AWS::WorkSpaces::Workspace
- AWS::WorkSpaces::WorkspacesPool

AWS::WorkSpaces::Workspace

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Available	Available	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
CPUUsage	CPUUsage	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ConnectionAttempt	ConnectionAttempt	Sum	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
ConnectionFailure	ConnectionFailure	Sum	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
ConnectionSuccess	ConnectionSuccess	Sum	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
InSessionLatency	InSessionLatency	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
Maintenance	Maintenance	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
MemoryUsage	MemoryUsage	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
Restoring	Restoring	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
RootVolumeDiskUsage	RootVolumeDiskUsage	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
SessionDisconnect	SessionDisconnect	Sum	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId
SessionLaunchTime	SessionLaunchTime	Histogram	AwsAccountId, BundleId, ComputeType, Protocol, RunningMode, WorkspaceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
Stopped	Stopped	Histogram	AwsAccountId ,BundleId, ComputeType ,Protocol, RunningMode , WorkspaceId
UDPPacketLossRate	UDPPacketLossRate	Histogram	AwsAccountId ,BundleId, ComputeType ,Protocol, RunningMode , WorkspaceId
Unhealthy	Unhealthy	Histogram	AwsAccountId ,BundleId, ComputeType ,Protocol, RunningMode , WorkspaceId
UpTime	UpTime	Histogram	AwsAccountId ,BundleId, ComputeType ,Protocol, RunningMode , WorkspaceId
UserConnected	UserConnected	Histogram	AwsAccountId ,BundleId, ComputeType ,Protocol, RunningMode , WorkspaceId

Metric	OpenTelemetry metric	Instrument type	Data point attributes
UserVolumeDiskUsage	UserVolumeDiskUsage	Histogram	AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId

AWS::WorkSpaces::WorkspacesPool

- cloud.resource_id enrichment: Supported
- tag.* enrichment: Not supported

Metric	OpenTelemetry metric	Instrument type	Data point attributes
ActiveUserSessionCapacity	ActiveUserSessionCapacity	Histogram	WorkSpaces pool ID
ActualUserSessionCapacity	ActualUserSessionCapacity	Histogram	WorkSpaces pool ID
AvailableUserSessionCapacity	AvailableUserSessionCapacity	Histogram	WorkSpaces pool ID
DesiredUserSessionCapacity	DesiredUserSessionCapacity	Histogram	WorkSpaces pool ID

Metric	OpenTelemetry metric	Instrument type	Data point attributes
InsufficientCapacityError	InsufficientCapacityError	Sum	WorkSpaces pool ID
PendingUserSessionCapacity	PendingUserSessionCapacity	Histogram	WorkSpaces pool ID
UserSessionsCapacityUtilization	UserSessionsCapacityUtilization	Histogram	WorkSpaces pool ID

Resource tags for telemetry

Use Amazon CloudWatch to enrich your AWS infrastructure metrics and CloudWatch Logs with AWS resource tags. You can set up comprehensive monitoring with CloudWatch metrics and alarms using tags. This helps monitor cloud infrastructure at scale by adapting alarms and metrics analysis as resources change.

To begin discovering and visualizing your telemetry by tags, you must first enable the resource tags for telemetry feature for your AWS account. When you enable this feature, Resource Explorer creates an AWS index and managed view that indexes and discovers resources and tags in your account. For more information, see Index in the Resource Explorer API reference guide and AWS managed views in the Resource Explorer user guide. CloudWatch uses this information to enrich your AWS infrastructure metrics and log events with related AWS resource tags. You can enable **resource tags for telemetry** at no additional cost.

Topics

- Enable resource tags on telemetry
- Using resource tags for telemetry
- Disable resource tags on telemetry
- Troubleshooting

Enable resource tags on telemetry

To add tag information to your telemetry data, enable resource tags for telemetry from the CloudWatch console, AWS SDK, CLI, or CloudFormation. The feature remains active until you turn it off. For more information, see Disable resource tags on telemetry.

Make sure you have permissions to enable resource tags for telemetry.

> **Note**
>
> To enable resource tags on telemetry, you must be signed in to an IAM principal that has the `observabilityadmin:StartTelemetryEnrichment`, `iam:CreateServiceLinkedRole`, `resource-explorer-2:CreateIndex`, `resource-explorer-2:CreateManagedView` and `resource-explorer-2:CreateStreamingAccessForService` permissions.

To enable resource tags for telemetry (CloudWatch Console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **CloudWatch**, then choose **Settings**.
3. In the **Enable resource tags for telemetry** pane, toggle the feature On.

> **Note**
>
> In the CloudWatch console, you must be signed in to an IAM principal that has the `observabilityadmin:GetTelemetryEnrichmentStatus` permission.

To enable resource tags for telemetry (AWS CLI)

Use the `start-telemetry-enrichment` command to enable resource tags for telemetry.

```
aws observabilityadmin start-telemetry-enrichment
```

After you complete these steps, CloudWatch begins enriching telemetry with tags.

To enable resource tags for telemetry (AWS CloudFormation)

You can enable resource tags for telemetry by adding an AWS::ObservabilityAdmin::TelemetryEnrichment resource to your CloudFormation template. Set the `Scope` property to `ACCOUNT` to enable the feature for your account.

Using resource tags for telemetry

Once you have enabled resource tags for telemetry, you can use this enriched data to create powerful monitoring solutions that scale with your infrastructure. Use tag-based queries to group and filter metrics across multiple resources, create dynamic alarms that automatically adapt to resource changes, and gain insights into your AWS environment organized by meaningful business or operational categories. This approach enables you to monitor resources by team, environment, application, or any other tagging strategy you use in your organization.

- **Creating tag-based Metrics Insights queries** – After you enable resource tags for telemetry in your account, you can create tag-based Metrics Insights queries to discover and visualize your AWS infrastructure metrics by tag. Example queries using tags can be seen in the CloudWatch Metrics Insights query builder documentation. *Monitoring accounts* can also make tag-based queries for metrics in *source accounts* which have enabled resource tags on their telemetry.
- **Creating tag-based CloudWatch alarms** – After you enable resource tags for telemetry in your account, you can create CloudWatch alarms on tag-based Metrics Insights queries to alert on your AWS infrastructure metrics by tag. Example alarms using tag-based queries can be seen in the CloudWatch Metric Insights alarms documentation.
- **Querying AWS metrics with PromQL using resource tags** – After you enable resource tags for telemetry in your account, you can enable OTel enrichment to query your AWS infrastructure metrics using PromQL with resource tags as labels. This lets you filter, aggregate, and alert on metrics by any tag you've applied to your AWS resources. For more information, see Enabling vended metrics in PromQL. For example queries using tags, see Querying vended AWS metrics with PromQL.
- **Creating tag-based CloudWatch Logs Insights queries** – After you enable resource tags for telemetry in your account, log events from supported resources are enriched with the associated resource tags. You can create tag-based Logs Insights queries to discover your log events by tag using facets such as `@aws.tag.env` or `@aws.tag.team`. For example: `filter @aws.tag.env = 'production'`. For more information, see the CloudWatch Logs Insights query syntax documentation.

Supported AWS resources

The following table lists the AWS resources that support resource tags for telemetry enrichment. When you enable resource tags for telemetry, CloudWatch can enrich metrics and log events from these resources with their associated resource tags.

For the specific metrics that CloudWatch enriches for each resource type, see Supported metrics for resource tags for telemetry.

Resource type	Metrics	Logs
AWS::APS::RuleGroupsNamespace	Yes	No
AWS::APS::Workspace	Yes	Yes
AWS::ApiGatewayV2::Api	Yes	No
AWS::AppFlow::Flow	Yes	No
AWS::AppSync::GraphQLApi	Yes	Yes
AWS::Athena::WorkGroup	Yes	No
AWS::Backup::BackupVault	Yes	No
AWS::CloudFront::Distribution	Yes	Yes
AWS::CloudFront::Function	Yes	Yes
AWS::CloudWatch::MetricStream	Yes	No
AWS::CodeGuruProfiler::ProfilingGroup	Yes	No
AWS::Cognito::UserPool	Yes	Yes
AWS::Connect::Instance	Yes	Yes
AWS::DAX::Cluster	Yes	No
AWS::DataSync::Task	Yes	No

Resource type	Metrics	Logs
AWS::DynamoDB::Table	Yes	No
AWS::EC2::CapacityReservation	Yes	No
AWS::EC2::ClientVpnEndpoint	Yes	No
AWS::EC2::Host	Yes	No
AWS::EC2::Instance	Yes	Yes
AWS::EC2::NatGateway	Yes	No
AWS::EC2::TransitGateway	Yes	Yes
AWS::EC2::VPC	Yes	Yes
AWS::EC2::VPNConnection	Yes	Yes
AWS::EC2::Volume	Yes	No
AWS::ECS::Cluster	Yes	Yes
AWS::ECS::Service	Yes	No
AWS::EFS::FileSystem	Yes	No
AWS::EKS::Cluster	Yes	Yes
AWS::EMR::Cluster	Yes	No
AWS::EMRServerless::Application	Yes	No
AWS::ElastiCache::CacheCluster	Yes	Yes
AWS::ElastiCache::ReplicationGroup	Yes	No
AWS::ElasticBeanstalk::Environment	Yes	No
AWS::ElasticLoadBalancing::LoadBalancer	Yes	No

Resource type	Metrics	Logs
AWS::ElasticLoadBalancingV2::LoadBalancer	Yes	Yes
AWS::ElasticLoadBalancingV2::TargetGroup	Yes	No
AWS::Events::Rule	Yes	No
AWS::FSx::FileSystem	Yes	No
AWS::FraudDetector::Detector	Yes	No
AWS::GameLift::GameSessionQueue	Yes	No
AWS::GameLift::MatchmakingConfiguration	Yes	No
AWS::Glue::Job	Yes	No
AWS::IVSChat::LoggingConfiguration	Yes	No
AWS::IoT::CACertificate	Yes	No
AWS::IoT::ScheduledAudit	Yes	No
AWS::IoT::SecurityProfile	Yes	No
AWS::IoT::TopicRule	Yes	No
AWS::KMS::Key	Yes	No
AWS::Kendra::DataSource	Yes	No
AWS::Kendra::Index	Yes	No
AWS::Kinesis::Stream	Yes	No
AWS::KinesisAnalyticsV2::Application	Yes	No

Resource type	Metrics	Logs
AWS::KinesisFirehose::DeliveryStream	Yes	No
AWS::Lambda::EventSourceMapping	Yes	No
AWS::Lambda::Function	Yes	Yes
AWS::MWAA::Environment	Yes	No
AWS::MediaTailor::Channel	Yes	No
AWS::MemoryDB::Cluster	Yes	No
AWS::NetworkFirewall::Firewall	Yes	Yes
AWS::OpenSearchServerless::Collection	Yes	No
AWS::OpenSearchService::Domain	Yes	No
AWS::Pipes::Pipe	Yes	Yes
AWS::RDS::DBCluster	Yes	Yes
AWS::RDS::DBInstance	Yes	No
AWS::RUM::AppMonitor	Yes	Yes
AWS::Redshift::Cluster	Yes	No
AWS::Route53::HealthCheck	Yes	No
AWS::Route53Resolver::FirewallRuleGroup	Yes	No
AWS::Route53Resolver::ResolverEndpoint	Yes	No
AWS::S3::Bucket	Yes	No
AWS::SNS::Topic	Yes	No
AWS::SQS::Queue	Yes	No

Resource type	Metrics	Logs
AWS::SageMaker::Endpoint	Yes	No
AWS::SageMaker::InferenceComponent	Yes	No
AWS::Scheduler::ScheduleGroup	Yes	No
AWS::Synthetics::Canary	Yes	No
AWS::Transfer::Connector	Yes	Yes
AWS::Transfer::Server	Yes	Yes
AWS::VpcLattice::Service	Yes	Yes
AWS::WorkSpaces::Workspace	Yes	No

Supported AWS Regions for resource tags for telemetry

The following table lists the AWS Regions where the resource tags for telemetry feature is available.

Region name	Region
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Taipei)	ap-east-2
Asia Pacific (Tokyo)	ap-northeast-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2

Region name	Region
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (New Zealand)	ap-southeast-6
Asia Pacific (Thailand)	ap-southeast-7
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
Europe (Frankfurt)	eu-central-1
Europe (Zurich)	eu-central-2
Europe (Stockholm)	eu-north-1
Europe (Milan)	eu-south-1
Europe (Spain)	eu-south-2
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Mexico (Central)	mx-central-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1

Region name	Region
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Supported metrics for resource tags for telemetry

The following sections list the metrics that CloudWatch enriches with AWS resource tags, grouped by namespace. For each namespace, the supported resource types are listed, with a table of metrics and the dimensions you can use with each.

CloudWatch treats each unique combination of dimensions as a separate metric. In the following tables, each bullet in the Dimensions column is a combination of dimensions that are published together.

AWS/AOSS

The AWS/AOSS namespace includes enriched metrics for the following resource types.

- AWS::OpenSearchServerless::Collection

AWS::OpenSearchServerless::Collection

Metric	Dimensions
2xx	• ClientId, CollectionId , CollectionName
3xx	• ClientId, CollectionId , CollectionName
4xx	• ClientId, CollectionId , CollectionName
5xx	• ClientId, CollectionId , CollectionName

Metric	Dimensions
ActiveCollection	• ClientId, CollectionId , CollectionName
ActiveCollection-Shadow	• ClientId, CollectionId , CollectionName
AutoOptimizeJob	• ClientId, CollectionId , CollectionName
IngestionDataRate-Shadow	• ClientId, CollectionId , CollectionName
IngestionDocumentErrors-Shadow	• ClientId, CollectionId , CollectionName
IngestionDocumentRate-Shadow	• ClientId, CollectionId , CollectionName
IngestionRequestErrors	• ClientId, CollectionId , CollectionName
IngestionRequestErrors-Shadow	• ClientId, CollectionId , CollectionName
IngestionRequestLatency	• ClientId, CollectionId , CollectionName
IngestionRequestRate	• ClientId, CollectionId , CollectionName
IngestionRequestRate-Shadow	• ClientId, CollectionId , CollectionName
IngestionRequestSuccess	• ClientId, CollectionId , CollectionName

Metric	Dimensions
IngestionRequestSuccess-Shadow	• ClientId, CollectionId , CollectionName
KNNRemoteIndexBuildFailureCount	• ClientId, CollectionId , CollectionName
KNNRemoteIndexBuildSuccessCount	• ClientId, CollectionId , CollectionName
KNNRemoteIndexBuildTime	• ClientId, CollectionId , CollectionName
SearchRequestErrors-Shadow	• ClientId, CollectionId , CollectionName
SearchRequestLatency	• ClientId, CollectionId , CollectionName
SearchRequestRate	• ClientId, CollectionId , CollectionName
SearchRequestRate-Shadow	• ClientId, CollectionId , CollectionName
SemanticSearchOCU	• AwsAccountId , CollectionId • ClientId, CollectionId , CollectionName

AWS/ApiGateway

The AWS/ApiGateway namespace includes enriched metrics for the following resource types.

- AWS::ApiGatewayV2::Api

AWS::ApiGatewayV2::Api

Metric	Dimensions
4xx	• ApiId • ApiId, Method, Resource, Stage
5xx	• ApiId • ApiId, Method, Resource, Stage
ClientError	• ApiId • ApiId, Route, Stage
ConnectCount	• ApiId • ApiId, Route, Stage
Count	• ApiId • ApiId, Method, Resource, Stage
DataProcessed	• ApiId • ApiId, Method, Resource, Stage
ExecutionError	• ApiId • ApiId, Route, Stage
IntegrationError	• ApiId • ApiId, Route, Stage
IntegrationLatency	• ApiId • ApiId, Method, Resource, Stage • ApiId, Route, Stage
Latency	• ApiId • ApiId, Method, Resource, Stage
MessageCount	• ApiId • ApiId, Route, Stage

AWS/AppFlow

The AWS/AppFlow namespace includes enriched metrics for the following resource types.

- AWS::AppFlow::Flow

AWS::AppFlow::Flow

Metric	Dimensions
FlowExecutionRecordsProcessed	• FlowName
FlowExecutionTime	• FlowName
FlowExecutionsFailed	• FlowName
FlowExecutionsStarted	• FlowName
FlowExecutionsSucceeded	• FlowName

AWS/AppSync

The AWS/AppSync namespace includes enriched metrics for the following resource types.

- AWS::AppSync::GraphQLApi

AWS::AppSync::GraphQLApi

Metric	Dimensions
4XXError	• GraphQLAPIId
5XXError	• GraphQLAPIId
ActiveConnections	• GraphQLAPIId
ActiveSubscriptions	• GraphQLAPIId
CacheHit	• GraphQLAPIId , Resolver

Metric	Dimensions
CacheMiss	• GraphQLAPIId ,Resolver
ConnectClientError	• GraphQLAPIId
ConnectRequests	• GraphQLAPIId
ConnectServerError	• GraphQLAPIId
ConnectSuccess	• GraphQLAPIId
ConnectionDuration	• GraphQLAPIId
DisconnectClientError	• GraphQLAPIId
DisconnectServerError	• GraphQLAPIId
DisconnectSuccess	• GraphQLAPIId
GraphQLError	• DataSource ,GraphQLAPIId • GraphQLAPIId ,Operation • GraphQLAPIId ,Resolver
InboundMessageDelayed	• GraphQLAPIId
InboundMessageDropped	• GraphQLAPIId
InboundMessageError	• GraphQLAPIId
InboundMessageFailure	• GraphQLAPIId
InboundMessageSuccess	• GraphQLAPIId
InvalidationRequestDropped	• GraphQLAPIId
InvalidationRequestError	• GraphQLAPIId
InvalidationRequestFailure	• GraphQLAPIId
InvalidationRequestSuccess	• GraphQLAPIId

Metric	Dimensions
InvalidationSuccess	• GraphQLAPIId
Latency	• DataSource ,GraphQLAPIId • GraphQLAPIId • GraphQLAPIId ,Resolver
OutboundMessages	• GraphQLAPIId
PublishDataMessageClientError	• GraphQLAPIId
PublishDataMessageServerError	• GraphQLAPIId
PublishDataMessageSize	• GraphQLAPIId
PublishDataMessageSuccess	• GraphQLAPIId
Requests	• DataSource ,GraphQLAPIId • GraphQLAPIId ,Operation • GraphQLAPIId ,Resolver
SubscribeClientError	• GraphQLAPIId
SubscribeServerError	• GraphQLAPIId
SubscribeSuccess	• GraphQLAPIId
UnsubscribeClientError	• GraphQLAPIId
UnsubscribeServerError	• GraphQLAPIId
UnsubscribeSuccess	• GraphQLAPIId

AWS/ApplicationELB

The AWS/ApplicationELB namespace includes enriched metrics for the following resource types.

- AWS::ElasticLoadBalancingV2::LoadBalancer

- AWS::ElasticLoadBalancingV2::TargetGroup

AWS::ElasticLoadBalancingV2::LoadBalancer

Metric	Dimensions
ActiveConnectionCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ActiveZonalShiftHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
AnomalousHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
AppCookieNonStickinessCount	• LoadBalancer
ClientTLSNegotiationErrorCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ConsumedLCUs	• LoadBalancer
DesyncMitigationMode_NonCompliant_Request_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
DroppedInvalidHeaderRequestCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ELBAuthError	• AvailabilityZone ,LoadBalancer • LoadBalancer
ELBAuthFailure	• AvailabilityZone ,LoadBalancer • LoadBalancer
ELBAuthLatency	• AvailabilityZone ,LoadBalancer

Metric	Dimensions
	• LoadBalancer
ELBAuthRefreshTokenSuccess	• AvailabilityZone ,LoadBalancer • LoadBalancer
ELBAuthSuccess	• AvailabilityZone ,LoadBalancer • LoadBalancer
ELBAuthUserClaimsSizeExceeded	• AvailabilityZone ,LoadBalancer • LoadBalancer
ExcessiveLowReputationPackets	• LoadBalancer
ForwardedInvalidHeaderRequestCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
GrpcRequestCount	• AvailabilityZone ,LoadBalancer ,TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
HTTPCode_ELB_3XX_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_ELB_4XX_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_ELB_500_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_ELB_502_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_ELB_503_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer

Metric	Dimensions
HTTPCode_ELB_504_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_ELB_5XX_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTPCode_Target_2XX_Count	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
HTTPCode_Target_3XX_Count	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
HTTPCode_Target_4XX_Count	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
HTTPCode_Target_5XX_Count	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
HTTP_Fixed_Response_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer

Metric	Dimensions
HTTP_Redirect_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HTTP_Redirect_Url_Limit_Exceeded_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
HealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
HealthyStateDNS	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
HealthyStateRouting	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
IPv6ProcessedBytes	• AvailabilityZone ,LoadBalancer • LoadBalancer
IPv6RequestCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
LambdaTargetProcessedBytes	• LoadBalancer
LambdaUserError	• LoadBalancer ,TargetGroup
LowReputationPacketsDropped	• AvailabilityZone ,LoadBalancer • LoadBalancer

Metric	Dimensions
LowReputationRequestsDenied	• AvailabilityZone ,LoadBalancer • LoadBalancer
MitigatedHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
NewConnectionCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
NonStickyRequestCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
PeakLCUs	• LoadBalancer
ProcessedBytes	• AvailabilityZone ,LoadBalancer • LoadBalancer
RejectedConnectionCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
RequestCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
RequestCountPerTarget	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
ReservedLCUs	• LoadBalancer
RuleEvaluations	• LoadBalancer

Metric	Dimensions
TargetConnectionErrorCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
TargetControlActiveChannelCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlChannelErrorCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlNewChannelCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlProcessedBytes	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlRequestCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlRequestRejectCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetControlWorkQueueLength	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetResponseTime	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup

Metric	Dimensions
TargetTLSNegotiationErrorCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
UnHealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
UnhealthyRoutingRequestCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
UnhealthyStateDNS	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
UnhealthyStateRouting	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup

AWS::ElasticLoadBalancingV2::TargetGroup

Metric	Dimensions
GrpcRequestCount	• AvailabilityZone ,TargetGroup • TargetGroup
LambdaInternalError	• TargetGroup

Metric	Dimensions
LambdaUserError	• TargetGroup
RequestCountPerTarget	• AvailabilityZone ,TargetGroup • TargetGroup

AWS/Athena

The AWS/Athena namespace includes enriched metrics for the following resource types.

- AWS::Athena::WorkGroup

AWS::Athena::WorkGroup

Metric	Dimensions
DPUConsumed	• WorkGroup
DPUCount	• QueryState ,QueryType ,WorkGroup
EngineExecutionTime	• QueryState ,QueryType ,WorkGroup
ProcessedBytes	• QueryState ,QueryType ,WorkGroup • WorkGroup
QueryPlanningTime	• QueryState ,QueryType ,WorkGroup
QueryQueueTime	• QueryState ,QueryType ,WorkGroup
ServicePreProcessingTime	• QueryState ,QueryType ,WorkGroup
ServiceProcessingTime	• QueryState ,QueryType ,WorkGroup
TotalExecutionTime	• QueryState ,QueryType ,WorkGroup

AWS/Backup

The AWS/Backup namespace includes enriched metrics for the following resource types.

- AWS::Backup::BackupVault

AWS::Backup::BackupVault

Metric	Dimensions
NumberOfBackupJobsCompleted	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsCreated	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsExpired	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsFailed	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsPartial	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsPending	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfBackupJobsRunning	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfCopyJobsCompleted	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfCopyJobsCreated	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfCopyJobsFailed	• BackupVaultName

Metric	Dimensions
	• BackupVaultName ,ResourceType
NumberOfCopyJobsRunning	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsCold	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsCompleted	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsCreating	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsDeleting	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsExpired	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRecoveryPointsPartial	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRestoreJobsCompleted	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRestoreJobsFailed	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRestoreJobsPending	• BackupVaultName • BackupVaultName ,ResourceType
NumberOfRestoreJobsRunning	• BackupVaultName • BackupVaultName ,ResourceType

AWS/ClientVPN

The AWS/ClientVPN namespace includes enriched metrics for the following resource types.

- AWS::EC2::ClientVpnEndpoint

AWS::EC2::ClientVpnEndpoint

Metric	Dimensions
ActiveConnectionsCount	• Endpoint
AuthenticationFailures	• Endpoint
ClientConnectHandlerDeniedConnections	• Endpoint
ClientConnectHandlerFailedServiceErrors	• Endpoint
ClientConnectHandlerInvalidResponses	• Endpoint
ClientConnectHandlerOtherExecutionErrors	• Endpoint
ClientConnectHandlerThrottlingErrors	• Endpoint
ClientConnectHandlerTimeouts	• Endpoint
CrlDaysToExpiry	• Endpoint
EgressBytes	• Endpoint
EgressPackets	• Endpoint
IngressBytes	• Endpoint
IngressPackets	• Endpoint

Metric	Dimensions
SelfServicePortalClientConfigurationDownloads	• Endpoint
ZeroHealthItemsCount	• Endpoint

AWS/CloudFront

The AWS/CloudFront namespace includes enriched metrics for the following resource types.

- AWS::CloudFront::Distribution
- AWS::CloudFront::Function

AWS::CloudFront::Distribution

Metric	Dimensions
401ErrorRate	• DistributionId ,Region
403ErrorRate	• DistributionId ,Region
404ErrorRate	• DistributionId ,Region
4xxErrorRate	• DistributionId ,Region
5xxErrorRate	• DistributionId ,Region
BytesDownloaded	• DistributionId ,Region
BytesUploaded	• DistributionId ,Region
CacheHitRate	• DistributionId ,Region
OriginLatency	• DistributionId ,Region
Requests	• DistributionId ,Region
TotalErrorRate	• DistributionId ,Region

AWS::CloudFront::Function

Metric	Dimensions
FunctionComputeUtilization	• DistributionId ,FunctionName , Region
FunctionExecutionErrors	• DistributionId ,FunctionName , Region
FunctionInvocations	• DistributionId ,FunctionName , Region
FunctionThrottles	• DistributionId ,FunctionName , Region
FunctionValidationErrors	• DistributionId ,FunctionName , Region
KvsReadErrors	• DistributionId ,FunctionName , Region
KvsReadRequests	• DistributionId ,FunctionName , Region

AWS/CloudWatch/MetricStreams

The AWS/CloudWatch/MetricStreams namespace includes enriched metrics for the following resource types.

- AWS::CloudWatch::MetricStream

AWS::CloudWatch::MetricStream

Metric	Dimensions
MetricUpdate	• MetricStreamName

Metric	Dimensions
PublishErrorRate	• MetricStreamName
TotalMetricUpdate	• MetricStreamName

AWS/CodeGuruProfiler

The AWS/CodeGuruProfiler namespace includes enriched metrics for the following resource types.

- AWS::CodeGuruProfiler::ProfilingGroup

AWS::CodeGuruProfiler::ProfilingGroup

Metric	Dimensions
Recommendations	• ProfilingGroupName

AWS/Cognito

The AWS/Cognito namespace includes enriched metrics for the following resource types.

- AWS::Cognito::UserPool

AWS::Cognito::UserPool

Metric	Dimensions
AccountTakeoverRisk	• Operation ,RiskLevel ,UserPoolId
CompromisedCredentialRisk	• Operation ,RiskLevel ,UserPoolId
FederationSuccesses	• IdentityProvider ,UserPool, UserPoolClient

Metric	Dimensions
FederationThrottles	• IdentityProvider ,UserPool, UserPoolClient
NoRisk	• Operation ,UserPoolId
Risk	• Operation ,UserPoolId
SignInSuccesses	• UserPool, UserPoolClient
SignInThrottles	• UserPool, UserPoolClient
SignUpSuccesses	• UserPool, UserPoolClient
SignUpThrottles	• UserPool, UserPoolClient
TokenRefreshSuccesses	• UserPool, UserPoolClient
TokenRefreshThrottles	• UserPool, UserPoolClient

AWS/Connect

The AWS/Connect namespace includes enriched metrics for the following resource types.

- AWS::Connect::Instance

AWS::Connect::Instance

Metric	Dimensions
CallRecordingUploadError	• InstanceId ,MetricGroup
CallsBreachingConcurrencyQuota	• InstanceId ,MetricGroup
CallsPerInterval	• InstanceId ,MetricGroup
ChatsBreachingActiveChatQuota	• InstanceId ,MetricGroup

Metric	Dimensions
ConcurrentActiveChats	• InstanceId ,MetricGroup
ConcurrentActiveChatsPercentage	• InstanceId ,MetricGroup
ConcurrentCalls	• InstanceId ,MetricGroup
ConcurrentCallsPercentage	• InstanceId ,MetricGroup
ConcurrentTasks	• InstanceId ,MetricGroup
ConcurrentTasksPercentage	• InstanceId ,MetricGroup
ContactFlowErrors	• ContactFlowName ,InstanceId , MetricGroup
LongestQueueWaitTime	• InstanceId ,MetricGroup , QueueName
MisconfiguredPhoneNumbers	• InstanceId ,MetricGroup
MissedCalls	• InstanceId ,MetricGroup
PublicSigningKeyUsage	• InstanceId ,SigningKeyId
QueueCapacityExceededError	• InstanceId ,MetricGroup , QueueName
QueueSize	• InstanceId ,MetricGroup , QueueName
SuccessfulChatsPerInterval	• InstanceId ,MetricGroup
TasksBreachingConcurrencyQuota	• InstanceId ,MetricGroup
ToInstancePacketLossRate	• InstanceId ,Participant ,Stream Type, Type of Connection

AWS/DAX

The AWS/DAX namespace includes enriched metrics for the following resource types.

- AWS::DAX::Cluster

AWS::DAX::Cluster

Metric	Dimensions
BaselineNetworkBytesInUtilization	• ClusterId • ClusterId ,NodeId
BaselineNetworkBytesOutUtilization	• ClusterId • ClusterId ,NodeId
BatchGetItemRequestCount	• ClusterId • ClusterId ,NodeId
BatchWriteItemRequestCount	• ClusterId • ClusterId ,NodeId
CPUCreditBalance	• ClusterId ,NodeId
CPUCreditUsage	• ClusterId • ClusterId ,NodeId
CPUSurplusCreditBalance	• ClusterId • ClusterId ,NodeId
CPUSurplusCreditsCharged	• ClusterId • ClusterId ,NodeId
CPUUtilization	• ClusterId • ClusterId ,NodeId
CacheMemoryUtilization	• ClusterId

Metric	Dimensions
	• ClusterId ,NodeId
ClientConnections	• ClusterId • ClusterId ,NodeId
DeleteItemRequestCount	• ClusterId • ClusterId ,NodeId
ErrorRequestCount	• ClusterId • ClusterId ,NodeId
EstimatedDbSize	• ClusterId • ClusterId ,NodeId
EvictedSize	• ClusterId • ClusterId ,NodeId
FailedRequestCount	• ClusterId • ClusterId ,NodeId
FaultRequestCount	• ClusterId • ClusterId ,NodeId
GetItemRequestCount	• ClusterId • ClusterId ,NodeId
ItemCacheHits	• ClusterId • ClusterId ,NodeId
ItemCacheMisses	• ClusterId • ClusterId ,NodeId
NetworkBytesIn	• ClusterId • ClusterId ,NodeId

Metric	Dimensions
NetworkBytesOut	• ClusterId • ClusterId ,NodeId
NetworkMaxBytesIn	• ClusterId • ClusterId ,NodeId
NetworkMaxBytesOut	• ClusterId • ClusterId ,NodeId
NetworkMaxPacketsIn	• ClusterId • ClusterId ,NodeId
NetworkMaxPacketsOut	• ClusterId • ClusterId ,NodeId
NetworkPacketsIn	• ClusterId • ClusterId ,NodeId
NetworkPacketsOut	• ClusterId • ClusterId ,NodeId
PutItemRequestCount	• ClusterId • ClusterId ,NodeId
QueryCacheHits	• ClusterId • ClusterId ,NodeId
QueryCacheMisses	• ClusterId • ClusterId ,NodeId
QueryRequestCount	• ClusterId • ClusterId ,NodeId
ScanCacheHits	• ClusterId • ClusterId ,NodeId

Metric	Dimensions
ScanCacheMisses	• ClusterId • ClusterId ,NodeId
ScanRequestCount	• ClusterId • ClusterId ,NodeId
ThrottledRequestCount	• ClusterId • ClusterId ,NodeId
TotalRequestCount	• ClusterId • ClusterId ,NodeId
TransactGetItemsCount	• ClusterId • ClusterId ,NodeId
TransactWriteItemsCount	• ClusterId • ClusterId ,NodeId
UpdateItemRequestCount	• ClusterId • ClusterId ,NodeId

AWS/DataSync

The AWS/DataSync namespace includes enriched metrics for the following resource types.

- AWS::DataSync::Task

AWS::DataSync::Task

Metric	Dimensions
BytesCompressed	• TaskId
BytesPreparedDestination	• TaskId

Metric	Dimensions
BytesPreparedSource	• TaskId
BytesTransferred	• TaskId
BytesVerifiedDestination	• TaskId
BytesVerifiedSource	• TaskId
BytesWritten	• TaskId
FilesDeleted	• TaskId
FilesListedSource	• TaskId
FilesPrepared	• TaskId
FilesPreparedDestination	• TaskId
FilesPreparedSource	• TaskId
FilesSkipped	• TaskId
FilesTransferred	• TaskId
FilesVerified	• TaskId
FilesVerifiedDestination	• TaskId
FilesVerifiedSource	• TaskId

AWS/DynamoDB

The AWS/DynamoDB namespace includes enriched metrics for the following resource types.

- AWS::DynamoDB::Table

AWS::DynamoDB::Table

Metric	Dimensions
AgeOfOldestUnreplicatedRecord	• DelegatedOperation ,TableName
ConditionalCheckFailedRequests	• TableName
ConsumedChangeDataCaptureUnits	• DelegatedOperation ,TableName
ConsumedReadCapacityUnits	• TableName
ConsumedWriteCapacityUnits	• GlobalSecondaryIndexName ,Source, TableName • Source, TableName • TableName
ProvisionedReadCapacityUnits	• TableName
ProvisionedWriteCapacityUnits	• TableName
ReadThrottleEvents	• TableName
ReturnedItemCount	• Operation ,TableName • Operation ,TableName ,Verb
SuccessfulRequestLatency	• Operation ,OperationType , TableName • Operation ,TableName • Operation ,TableName ,Verb
SystemErrors	• Operation ,TableName
ThrottledPutRecordCount	• DelegatedOperation ,TableName
ThrottledRequests	• Operation ,TableName • Operation ,TableName ,Verb
TimeToLiveDeletedItemCount	• TableName

Metric	Dimensions
TransactionConflict	• TableName
WriteThrottleEvents	• TableName

AWS/EBS

The AWS/EBS namespace includes enriched metrics for the following resource types.

- AWS::EC2::Volume

AWS::EC2::Volume

Metric	Dimensions
BurstBalance	• VolumeId
VolumeAvgIOPS	• InstanceId , VolumeId
VolumeAvgReadLatency	• InstanceId , VolumeId
VolumeAvgThroughput	• InstanceId , VolumeId
VolumeAvgWriteLatency	• InstanceId , VolumeId
VolumeConsumedReadWriteOps	• VolumeId
VolumeIOPSExceededCheck	• InstanceId , VolumeId
VolumeIdleTime	• VolumeId
VolumeQueueLength	• VolumeId
VolumeReadBytes	• VolumeId
VolumeReadOps	• VolumeId
VolumeStalledIOCheck	• InstanceId , VolumeId

Metric	Dimensions
VolumeThroughputExceededCheck	• InstanceId , VolumeId
VolumeThroughputPercentage	• VolumeId
VolumeTotalReadTime	• VolumeId
VolumeTotalWriteTime	• VolumeId
VolumeWriteBytes	• VolumeId
VolumeWriteOps	• VolumeId

AWS/EC2

The AWS/EC2 namespace includes enriched metrics for the following resource types.

- AWS::EC2::Host
- AWS::EC2::Instance
- AWS::EC2::VPC

AWS::EC2::Host

Metric	Dimensions
DedicatedHostCPUUtilization	• HostId

AWS::EC2::Instance

Metric	Dimensions
CPUCreditBalance	• InstanceId
CPUCreditUsage	• InstanceId
CPUSurplusCreditBalance	• InstanceId

Metric	Dimensions
CPUSurplusCreditsCharged	• InstanceId
CPUUtilization	• InstanceId
DiskReadBytes	• InstanceId
DiskReadOps	• InstanceId
DiskWriteBytes	• InstanceId
DiskWriteOps	• InstanceId
EBSByteBalance%	• InstanceId
EBSIOBalance%	• InstanceId
EBSReadBytes	• InstanceId
EBSReadOps	• InstanceId
EBSWriteBytes	• InstanceId
EBSWriteOps	• InstanceId
GPUPowerUtilization	• InstanceId
InstanceEBSIOPSExceededCheck	• InstanceId
InstanceEBSThroughputExceeded Check	• InstanceId
MetadataNoToken	• InstanceId
MetadataNoTokenRejected	• InstanceId
NetworkIn	• InstanceId
NetworkMirrorIn	• InstanceId
NetworkMirrorOut	• InstanceId

Metric	Dimensions
NetworkOut	• InstanceId
NetworkPacketsIn	• InstanceId
NetworkPacketsMirrorIn	• InstanceId
NetworkPacketsMirrorOut	• InstanceId
NetworkPacketsOut	• InstanceId
NetworkPacketsSkipMirrorIn	• InstanceId
NetworkPacketsSkipMirrorOut	• InstanceId
NetworkSkipMirrorIn	• InstanceId
NetworkSkipMirrorOut	• InstanceId
StatusCheckFailed	• InstanceId
StatusCheckFailed_Application	• InstanceId
StatusCheckFailed_AttachedEBS	• InstanceId
StatusCheckFailed_Instance	• InstanceId
StatusCheckFailed_System	• InstanceId

AWS::EC2::VPC

Metric	Dimensions
NetworkAddressUsage	• Per-VPC Metrics
NetworkAddressUsagePeered	• Per-VPC Metrics

AWS/EC2CapacityReservations

The AWS/EC2CapacityReservations namespace includes enriched metrics for the following resource types.

- AWS::EC2::CapacityReservation

AWS::EC2::CapacityReservation

Metric	Dimensions
AvailableInstanceCount	• CapacityReservationId
InstanceUtilization	• CapacityReservationId
TotalInstanceCount	• CapacityReservationId
UsedInstanceCount	• CapacityReservationId

AWS/ECS

The AWS/ECS namespace includes enriched metrics for the following resource types.

- AWS::ECS::Cluster
- AWS::ECS::Service

AWS::ECS::Cluster

Metric	Dimensions
CPUReservation	• ClusterName
CPUUtilization	• ClusterName
GPUReservation	• ClusterName
MemoryReservation	• ClusterName

Metric	Dimensions
MemoryUtilization	• ClusterName

AWS::ECS::Service

Metric	Dimensions
CPUUtilization	• ClusterName , ServiceName
MemoryUtilization	• ClusterName , ServiceName

AWS/EFS

The AWS/EFS namespace includes enriched metrics for the following resource types.

- AWS::EFS::FileSystem

AWS::EFS::FileSystem

Metric	Dimensions
BurstCreditBalance	• FileSystemId
ClientConnections	• FileSystemId
DataReadIOBytes	• FileSystemId
DataWriteIOBytes	• FileSystemId
MetadataIOBytes	• FileSystemId
MeteredIOBytes	• FileSystemId
PercentIOLimit	• FileSystemId
PermittedThroughput	• FileSystemId

Metric	Dimensions
StorageBytes	• FileSystemId , StorageClass
TimeSinceLastSync	• DestinationFileSystemId , FileSystemId
TotalIOBytes	• FileSystemId

AWS/EKS

The AWS/EKS namespace includes enriched metrics for the following resource types.

- AWS::EKS::Cluster

AWS::EKS::Cluster

Metric	Dimensions
apiserver_admission_webhook_rejection_count_ADMIT	• ClusterName
apiserver_admission_webhook_rejection_count_VALIDATING	• ClusterName
apiserver_admission_webhook_request_total	• ClusterName
apiserver_admission_webhook_request_total_ADMIT	• ClusterName
apiserver_admission_webhook_request_total_VALIDATING	• ClusterName
apiserver_current_inflight_requests_MUTATING	• ClusterName

Metric	Dimensions
apiserver_current_inflight_requests_READONLY	• ClusterName
apiserver_flowcontrol_current_executing_seats	• ClusterName
apiserver_request_duration_seconds_DELETE_P99	• ClusterName
apiserver_request_duration_seconds_GET_P99	• ClusterName
apiserver_request_duration_seconds_LIST_P99	• ClusterName
apiserver_request_duration_seconds_PATCH_P99	• ClusterName
apiserver_request_duration_seconds_POST_P99	• ClusterName
apiserver_request_duration_seconds_PUT_P99	• ClusterName
apiserver_request_total	• ClusterName
apiserver_request_total_429	• ClusterName
apiserver_request_total_4XX	• ClusterName
apiserver_request_total_5XX	• ClusterName
apiserver_request_total_LIST_PODS	• ClusterName
apiserver_storage_size_bytes	• ClusterName

Metric	Dimensions
etcd_mvcc_db_total_size_in_use_in_bytes	• ClusterName
scheduler_pending_pods	• ClusterName
scheduler_pending_pods_ACTIVEQ	• ClusterName
scheduler_pending_pods_BACKOFF	• ClusterName
scheduler_pending_pods_GATED	• ClusterName
scheduler_pending_pods_UNSCHEDULABLE	• ClusterName
scheduler_schedule_attempts_ERROR	• ClusterName
scheduler_schedule_attempts_SCHEDULED	• ClusterName
scheduler_schedule_attempts_UNSCHEDULABLE	• ClusterName
scheduler_schedule_attempts_total	• ClusterName

AWS/ELB

The AWS/ELB namespace includes enriched metrics for the following resource types.

- AWS::ElasticLoadBalancing::LoadBalancer

AWS::ElasticLoadBalancing::LoadBalancer

Metric	Dimensions
BackendConnectionErrors	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
DesyncMitigationMode_NonCompliant_Request_Count	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
EstimatedALBActiveConnectionCount	• LoadBalancerName
EstimatedALBNewConnectionCount	• LoadBalancerName
EstimatedProcessedBytes	• LoadBalancerName
HTTPCode_Backend_2XX	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
HTTPCode_Backend_3XX	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
HTTPCode_Backend_4XX	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
HTTPCode_Backend_5XX	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
HTTPCode_ELB_4XX	• AvailabilityZone ,LoadBalancerName

Metric	Dimensions
	• LoadBalancerName
HTTPCode_ELB_5XX	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
HealthyHostCount	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
Latency	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
RequestCount	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
SpilloverCount	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
SurgeQueueLength	• AvailabilityZone ,LoadBalancerName • LoadBalancerName
UnHealthyHostCount	• AvailabilityZone ,LoadBalancerName • LoadBalancerName

AWS/EMRServerless

The AWS/EMRServerless namespace includes enriched metrics for the following resource types.

- AWS::EMRServerless::Application

AWS::EMRServerless::Application

Metric	Dimensions
CPUAllocated	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
CancelledJobs	• ApplicationId • ApplicationId ,ApplicationName
CancellingJobs	• ApplicationId • ApplicationId ,ApplicationName
FailedJobs	• ApplicationId • ApplicationId ,ApplicationName
IdleWorkerCount	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
MaxCPUAllowed	• ApplicationId • ApplicationId ,ApplicationName
MaxMemoryAllowed	• ApplicationId • ApplicationId ,ApplicationName
MaxStorageAllowed	• ApplicationId

Metric	Dimensions
	• ApplicationId ,ApplicationName
MemoryAllocated	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
PendingCreationWorkerCount	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
PendingJobs	• ApplicationId • ApplicationId ,ApplicationName
RunningJobs	• ApplicationId • ApplicationId ,ApplicationName
RunningWorkerCount	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
ScheduledJobs	• ApplicationId • ApplicationId ,ApplicationName

Metric	Dimensions
StorageAllocated	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
SubmittedJobs	• ApplicationId • ApplicationId ,ApplicationName
SuccessJobs	• ApplicationId • ApplicationId ,ApplicationName
TotalWorkerCount	• ApplicationId • ApplicationId ,ApplicationName • ApplicationId ,ApplicationName , CapacityAllocationType , WorkerType • ApplicationId ,CapacityA llocationType ,WorkerType
WorkerCpuAllocated	• ApplicationId ,ApplicationName , CapacityAllocationType ,JobId, JobName, WorkerType • ApplicationId ,CapacityA llocationType ,JobId, WorkerType
WorkerCpuUsed	• ApplicationId ,ApplicationName , CapacityAllocationType ,JobId, JobName, WorkerType • ApplicationId ,CapacityA llocationType ,JobId, WorkerType

Metric	Dimensions
WorkerEphemeralStorageAllocated	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityA llocationType , JobId, WorkerType
WorkerEphemeralStorageUsed	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityA llocationType , JobId, WorkerType
WorkerMemoryAllocated	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityA llocationType , JobId, WorkerType
WorkerMemoryUsed	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityA llocationType , JobId, WorkerType
WorkerStorageReadBytes	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityA llocationType , JobId, WorkerType

Metric	Dimensions
WorkerStorageWriteBytes	• ApplicationId , ApplicationName , CapacityAllocationType , JobId, JobName, WorkerType • ApplicationId , CapacityAllocationType , JobId, WorkerType

AWS/ES

The AWS/ES namespace includes enriched metrics for the following resource types.

- AWS::OpenSearchService::Domain

AWS::OpenSearchService::Domain

Metric	Dimensions
2xx	• ClientId, DomainName
3xx	• ClientId, DomainName
4xx	• ClientId, DomainName
5xx	• ClientId, DomainName
ADAnomalyDetectorsIndexStatus.red	• ClientId, DomainName
ADAnomalyDetectorsIndexStatusIndexExists	• ClientId, DomainName
ADAnomalyResultsIndexStatus.red	• ClientId, DomainName
ADAnomalyResultsIndexStatusIndexExists	• ClientId, DomainName
ADExecuteFailureCount	• ClientId, DomainName

Metric	Dimensions
	• ClientId, DomainName , NodeId
ADExecuteRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ADHCExecuteFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ADHCExecuteRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ADModelsCheckpointIndexStatus.red	• ClientId, DomainName
ADModelsCheckpointIndexStatusIndexExists	• ClientId, DomainName
ADPluginUnhealthy	• ClientId, DomainName
ActiveDataNode	• ClientId, DomainName , NodeId
AlertingDegraded	• ClientId, DomainName
AlertingIndexExists	• ClientId, DomainName
AlertingIndexStatus.green	• ClientId, DomainName
AlertingIndexStatus.red	• ClientId, DomainName
AlertingIndexStatus.yellow	• ClientId, DomainName
AlertingNodesNotOnSchedule	• ClientId, DomainName
AlertingNodesOnSchedule	• ClientId, DomainName
AlertingScheduledJobEnabled	• ClientId, DomainName

Metric	Dimensions
AsyncQueryCancelApiFailedRequestCusErrCount	• ClientId, DomainName
AsyncQueryCancelApiFailedRequestSysErrCount	• ClientId, DomainName
AsyncQueryCancelApiRequestCount	• ClientId, DomainName
AsyncQueryCreateApiFailedRequestCusErrCount	• ClientId, DomainName
AsyncQueryCreateApiFailedRequestSysErrCount	• ClientId, DomainName
AsyncQueryCreateApiRequestCount	• ClientId, DomainName
AsyncQueryGetApiFailedRequestCusErrCount	• ClientId, DomainName
AsyncQueryGetApiFailedRequestSysErrCount	• ClientId, DomainName
AsyncQueryGetApiRequestCount	• ClientId, DomainName
AsynchronousSearchCancelled	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchCompletionRate	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchFailureRate	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchInitializedRate	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
AsynchronousSearchPersistFailedRate	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchPersistRate	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchRunningCurrent	• ClientId, DomainName • ClientId, DomainName , NodeId
AsynchronousSearchStoreHealthRed	• ClientId, DomainName
AsynchronousSearchStoreSize	• ClientId, DomainName
AsynchronousSearchStoredResponseCount	• ClientId, DomainName
AsynchronousSearchSubmissionRate	• ClientId, DomainName • ClientId, DomainName , NodeId
AutoFollowLeaderCallFailure	• ClientId, ConnectionId , DomainName • ClientId, DomainName • ClientId, DomainName , LeaderAlias
AutoFollowNumFailedStartReplication	• ClientId, ConnectionId , DomainName • ClientId, DomainName • ClientId, DomainName , LeaderAlias

Metric	Dimensions
AutoFollowNumSuccessStartReplication	• ClientId, ConnectionId , DomainName • ClientId, DomainName • ClientId, DomainName , LeaderAlias
AutoTuneChangesHistoryHeapSize	• AutotuningType , ClientId, DomainName , TargetId
AutoTuneFailed	• AutotuningType , ClientId, DomainName , TargetId
AutoTuneSucceeded	• AutotuningType , ClientId, DomainName , TargetId
AutoTuneValue	• AutotuningType , ClientId, DomainName , TargetId
AutomatedSnapshotFailure	• ClientId, DomainName
AvgPointInTimeAliveTime	• ClientId, DomainName
BurstBalance	• ClientId, DomainName • ClientId, DomainName , NodeId
CPUCreditBalance	• ClientId, DomainName
CPUUtilization	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
CanMatchCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId

Metric	Dimensions
CanMatchTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
CanMatchTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
ClusterIndexWritesBlocked	• ClientId, DomainName
ClusterStatus.green	• ClientId, DomainName
ClusterStatus.red	• ClientId, DomainName
ClusterStatus.yellow	• ClientId, DomainName
ClusterUsedSpace	• ClientId, DomainName
ColdStorageSpaceUtilization	• ClientId, DomainName
ColdToWarmMigrationFailureCount	• ClientId, DomainName
ColdToWarmMigrationLatency	• ClientId, DomainName
ColdToWarmMigrationQueueSize	• ClientId, DomainName
ColdToWarmMigrationSuccessCount	• ClientId, DomainName
ComponentTemplateCount	• ClientId, DomainName
ConcurrentSearchLatency	• ClientId, DomainName • ClientId, DomainName , NodeId
ConcurrentSearchRate	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
CoordinatingWriteRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorCPUUtilization	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorFreeStorageSpace	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorJVMGCOldCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorJVMGCOldCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorJVMGCYoungCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorJVMGCYoungCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorJVMMemoryPressure	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorOldGenJVMMemoryPressure	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorSysMemoryUtilization	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
CoordinatorThreadpoolForce_mergeQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolForce_mergeRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolForce_mergeThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolIndexQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolIndexRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolIndexThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolOpendistro_monitor_runnerQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolOpendistro_monitor_runnerRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolOpendistro_monitor_runnerThreads	• ClientId, DomainName , NodeId
CoordinatorThreadpoolSearchQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolSearchRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolSearchThreads	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
CoordinatorThreadpoolWriteQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolWriteRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolWriteThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolsql-workerQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolsql-workerRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
CoordinatorThreadpoolsql-workerThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
CrossClusterInboundReplicationRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CrossClusterInboundRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CrossClusterInboundSearchRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CrossClusterOutboundConnections	• ClientId, ConnectionId , DomainName • ClientId, DomainName

Metric	Dimensions
CrossClusterOutboundReplicationRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CrossClusterOutboundRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CrossClusterOutboundSearchRequests	• ClientId, ConnectionId , DomainName • ClientId, DomainName
CurrentPointInTime	• ClientId, DomainName , NodeId
DataNodes	• ActiveAZ, ClientId, DomainName
DataNodesShards.active	• ActiveAZ, ClientId, DomainName
DataNodesShards.initialising	• ActiveAZ, ClientId, DomainName
DataNodesShards.relocating	• ActiveAZ, ClientId, DomainName
DataNodesShards.unassigned	• ActiveAZ, ClientId, DomainName
DeletedDocuments	• ClientId, DomainName
DfsPreQueryCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
DfsPreQueryTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId

Metric	Dimensions
DfsPreQueryTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
DfsQueryCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
DfsQueryTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
DfsQueryTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
DiskQueueDepth	• ClientId, DomainName
ESReportingFailedRequestSysErrCount	• ClientId, DomainName
ESReportingFailedRequestUserErrCount	• ClientId, DomainName
ESReportingRequestCount	• ClientId, DomainName
ESReportingSuccessCount	• ClientId, DomainName
ElasticsearchRequests	• ClientId, DomainName
EnforcedWorkloadGroupCount	• ClientId, DomainName
ExpandCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId

Metric	Dimensions
ExpandTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
ExpandTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
FetchCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
FetchTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
FetchTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
FollowerCheckPoint	• ClientId, ConnectionId , DomainNam e • ClientId, DomainName , LeaderAlias
ForecastCheckpointIndexStat us.red	• ClientId, DomainName
ForecastCheckpointIndexStat usIndexExists	• ClientId, DomainName
ForecastConfigIndexStatus.red	• ClientId, DomainName
ForecastConfigIndexStatusIn dexExists	• ClientId, DomainName

Metric	Dimensions
ForecastExecuteFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ForecastExecuteRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ForecastHCExecuteFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ForecastHCExecuteRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ForecastModelCorruptionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
ForecastResultsIndexStatus.red	• ClientId, DomainName
ForecastResultsIndexStatusIndexExists	• ClientId, DomainName
ForecastStateIndexStatus.red	• ClientId, DomainName
ForecastStateIndexStatusIndexExists	• ClientId, DomainName
ForecastUnhealthy	• ClientId, DomainName
ForecasterCount	• ClientId, DomainName
FreeStorageSpace	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
HCForecasterCount	• ClientId, DomainName
HasActivePointInTime	• ClientId, DomainName

Metric	Dimensions
HasUsedPointInTime	• ClientId, DomainName
HighSwapUsage	• ClientId, DomainName , NodeId
HotStorageSpaceUtilization	• ClientId, DomainName
HotToWarmMigrationFailureCount	• ClientId, DomainName
HotToWarmMigrationForceMergeLatency	• ClientId, DomainName
HotToWarmMigrationProcessingLatency	• ClientId, DomainName
HotToWarmMigrationQueueSize	• ClientId, DomainName
HotToWarmMigrationSnapshotLatency	• ClientId, DomainName
HotToWarmMigrationSuccessCount	• ClientId, DomainName
HotToWarmMigrationSuccessLatency	• ClientId, DomainName
ISMPolicyCount	• ClientId, DomainName
InFlightFetches	• ClientId, DomainName
IndexingLatency	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
IndexingRate	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
InvalidHostHeaderRequests	• ClientId, DomainName

Metric	Dimensions
IopsThrottle	• ClientId, DomainName • ClientId, DomainName , NodeId
JVMGCOldCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
JVMGCOldCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
JVMGCYoungCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
JVMGCYoungCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
JVMMemoryPressure	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
KMSKeyError	• ClientId, DomainName
KMSKeyInaccessible	• ClientId, DomainName
KNNCacheCapacityReached	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNCircuitBreakerTriggered	• ClientId, DomainName
KNNEvictionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNFaissInitialized	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
KNNGraphIndexErrors	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNGraphIndexRequests	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNGraphMemoryUsage	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNGraphMemoryUsagePercentage	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNGraphQueryErrors	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNGraphQueryRequests	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNHitCount	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNLoadExceptionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNLoadSuccessCount	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNLuceneInitialized	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNMissCount	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNNmslibInitialized	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
KNNQueryRequests	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNRemoteBuildEnabled	• ClientId, DomainName
KNNRemoteIndexBuildFailureCount	• ClientId, DomainName
KNNRemoteIndexBuildSuccessCount	• ClientId, DomainName
KNNScriptCompilationErrors	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNScriptCompilations	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNScriptQueryErrors	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNScriptQueryRequests	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNTotalLoadTime	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNTrainingErrors	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNTrainingMemoryUsage	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNTrainingMemoryUsagePercentage	• ClientId, DomainName • ClientId, DomainName , NodeId
KNNTrainingRequests	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
KibanaConcurrentConnections	• ClientId, DomainName , NodeId
KibanaHealthyNode	• ClientId, DomainName , NodeId
KibanaHealthyNodes	• ClientId, DomainName
KibanaHeapTotal	• ClientId, DomainName , NodeId
KibanaHeapUsed	• ClientId, DomainName , NodeId
KibanaHeapUtilization	• ClientId, DomainName , NodeId
KibanaIndexMigrationFailed	• ClientId, DomainName
KibanaOS1MinuteLoad	• ClientId, DomainName , NodeId
KibanaRequestTotal	• ClientId, DomainName , NodeId
KibanaResponseTimesMaxInMillis	• ClientId, DomainName , NodeId
LTRFeatureMemoryUsageInBytes	• ClientId, DomainName • ClientId, DomainName , NodeId
LTRFeaturesetMemoryUsageInBytes	• ClientId, DomainName • ClientId, DomainName , NodeId
LTRModelMemoryUsageInBytes	• ClientId, DomainName • ClientId, DomainName , NodeId
LTRPluginUnhealthy	• ClientId, DomainName
LTRRequestErrorCount	• ClientId, DomainName • ClientId, DomainName , NodeId
LTRRequestTotalCount	• ClientId, DomainName • ClientId, DomainName , NodeId
LTRStatus.red	• ClientId, DomainName

Metric	Dimensions
LeaderCheckPoint	• ClientId, ConnectionId , DomainName • ClientId, DomainName , LeaderAlias
MLCircuitBreakerTriggerCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MLCommonsPluginUnhealthy	• ClientId, DomainName
MLConnectorCount	• ClientId, DomainName
MLConnectorIndexStatus.red	• ClientId, DomainName
MLConnectorIndexStatusIndex Exists	• ClientId, DomainName
MLDeployedModelCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MLExecutingTaskCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MLFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MLModelCount	• ClientId, DomainName
MLModelIndexStatus.red	• ClientId, DomainName
MLModelIndexStatusIndexExists	• ClientId, DomainName
MLRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MLTaskIndexStatus.red	• ClientId, DomainName
MLTaskIndexStatusIndexExists	• ClientId, DomainName

Metric	Dimensions
MasterCPUCreditBalance	• ClientId, DomainName
MasterCPUUtilization	• ClientId, DomainName • ClientId, DomainName , NodeId
MasterFreeStorageSpace	• ClientId, DomainName • ClientId, DomainName , NodeId
MasterJVMMemoryPressure	• ClientId, DomainName • ClientId, DomainName , NodeId
MasterOldGenJVMMemoryPressure	• ClientId, DomainName • ClientId, DomainName , NodeId
MasterReachableFromNode	• ClientId, DomainName
MasterSysMemoryUtilization	• ClientId, DomainName • ClientId, DomainName , NodeId
MaxProvisionedThroughput	• ClientId, DomainName • ClientId, DomainName , NodeId
MlCircuitBreakerTriggerCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlConnectorCount	• ClientId, DomainName
MlConnectorIndexStatus.red	• ClientId, DomainName
MlConnectorIndexStatusIndex Exists	• ClientId, DomainName
MlDeployedModelCount	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
MlExecutingTaskCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlModelCount	• ClientId, DomainName
MlModelIndexStatus.red	• ClientId, DomainName
MlModelIndexStatusIndexExists	• ClientId, DomainName
MlNodeExecutingTaskCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlNodeTotalCircuitBreakerTriggerCount	• ClientId, DomainName , NodeId
MlNodeTotalFailureCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlNodeTotalModelCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlNodeTotalRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlRequestCount	• ClientId, DomainName • ClientId, DomainName , NodeId
MlTaskIndexStatus.red	• ClientId, DomainName
MlTaskIndexStatusIndexExists	• ClientId, DomainName
Nodes	• ClientId, DomainName

Metric	Dimensions
OldGenJVMMemoryPressure	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
OpenContexts	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
OpenContexts-BeagleStone	• ClientId, DomainName
OpenSearchDashboardsConcurrentConnections	• ClientId, DomainName , NodeId
OpenSearchDashboardsHealthyNode	• ClientId, DomainName , NodeId
OpenSearchDashboardsHealthyNodes	• ClientId, DomainName
OpenSearchDashboardsHeapTotal	• ClientId, DomainName , NodeId
OpenSearchDashboardsHeapUsed	• ClientId, DomainName , NodeId
OpenSearchDashboardsHeapUtilization	• ClientId, DomainName , NodeId
OpenSearchDashboardsIndexMigrationFailed	• ClientId, DomainName
OpenSearchDashboardsOS1MinuteLoad	• ClientId, DomainName , NodeId
OpenSearchDashboardsRequestTotal	• ClientId, DomainName , NodeId
OpenSearchDashboardsResponseTimesMaxInMillis	• ClientId, DomainName , NodeId
OpenSearchRequests	• ClientId, DomainName

Metric	Dimensions
OpensearchDashboardsReportingFailedRequestSysErrCount	• ClientId, DomainName
OpensearchDashboardsReportingFailedRequestUserErrCount	• ClientId, DomainName
OpensearchDashboardsReportingRequestCount	• ClientId, DomainName
OpensearchDashboardsReportingSuccessCount	• ClientId, DomainName
PPLFailedRequestCountByCusErr	• ClientId, DomainName
PPLFailedRequestCountBySysErr	• ClientId, DomainName
PPLRequestCount	• ClientId, DomainName
PrimaryWriteRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
QueryCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
QueryFailure	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
QueryFailure-BeagleStone	• ClientId, DomainName
QuerySuccess	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
QuerySuccess-BeagleStone	• ClientId, DomainName

Metric	Dimensions
QueryTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
QueryTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
ReadIOPS	• ClientId, DomainName • ClientId, DomainName , NodeId
ReadIOPSMicroBursting	• ClientId, DomainName • ClientId, DomainName , NodeId
ReadLatency	• ClientId, DomainName • ClientId, DomainName , NodeId
ReadThroughput	• ClientId, DomainName • ClientId, DomainName , NodeId
ReadThroughputMicroBursting	• ClientId, DomainName • ClientId, DomainName , NodeId
RemoteStorageUsedSpace	• ClientId, DomainName
RemoteStorageWriteRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ReplicaWriteRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ReplicationLagMaxTime	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
ReplicationNumBootstrapping Indices	• ClientId, DomainName
ReplicationNumFailedIndices	• ClientId, DomainName
ReplicationNumIndexTasks	• ClientId, DomainName
ReplicationNumPausedIndices	• ClientId, DomainName
ReplicationNumShardTasks	• ClientId, DomainName
ReplicationNumSyncingIndices	• ClientId, DomainName
ReplicationRate	• ClientId, ConnectionId , DomainNam e • ClientId, DomainName , LeaderAlias
SQLDefaultCursorRequestCount	• ClientId, DomainName
SQLFailedRequestCountByCusErr	• ClientId, DomainName
SQLFailedRequestCountBySysErr	• ClientId, DomainName
SQLRequestCount	• ClientId, DomainName
SQLUnhealthy	• ClientId, DomainName
ScrollCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
ScrollCurrent-BeagleStone	• ClientId, DomainName
ScrollTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId

Metric	Dimensions
ScrollTotal-BeagleStone	• ClientId, DomainName
SearchIdleReactivateCountTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
SearchLatency	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
SearchPipelineRequestFailure	• ClientId, DomainName • ClientId, DomainName , Processor
SearchPipelineRequestTime	• ClientId, DomainName • ClientId, DomainName , Processor
SearchPipelineUnhealthy	• ClientId, DomainName
SearchRate	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
SearchShardTaskCancelled	• ClientId, DomainName • ClientId, DomainName , NodeId
SearchTaskCancelled	• ClientId, DomainName • ClientId, DomainName , NodeId
SearchableDocuments	• ClientId, DomainName
SegmentCount	• ClientId, DomainName , NodeId

Metric	Dimensions
ShardCount	• ActiveAZ, ClientId, DomainName , NodeId, ShardRole • ActiveAZ, ClientId, DomainName , ShardRole • ClientId, DomainName , NodeId, ShardRole
Shards.active	• ClientId, DomainName
Shards.activePrimary	• ClientId, DomainName
Shards.delayedUnassigned	• ClientId, DomainName
Shards.initializing	• ClientId, DomainName
Shards.relocating	• ClientId, DomainName
Shards.unassigned	• ClientId, DomainName
SingleStreamForecasterCount	• ClientId, DomainName
SnapshotCount	• ClientId, DomainName , Repository
SnapshotFailures	• ClientId, DomainName , Repository
SoftWorkloadGroupCount	• ClientId, DomainName
SysMemoryUtilization	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
TLSNegotiationError	• ClientId, DomainName
ThreadCount	• ClientId, DomainName
ThreadpoolBulkQueue	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
ThreadpoolBulkRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolBulkThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolForce_mergeQueue	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolForce_mergeRejected	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolForce_mergeThreads	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolIndexQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolIndexRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolIndexSearcherQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolIndexSearcherRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolIndexSearcherThreads	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
ThreadpoolIndexThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolMergeQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolMergeRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolMergeThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolOpendistro_monitor_runnerQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolOpendistro_monitor_runnerRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolOpendistro_monitor_runnerThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolSearchQueue	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolSearchRejected	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolSearchThreads	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
ThreadpoolWriteQueue	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolWriteRejected	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThreadpoolWriteThreads	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
Threadpoolsql-workerQueue	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
Threadpoolsql-workerRejected	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
Threadpoolsql-workerThreads	• ActiveAZ, ClientId, DomainName • ClientId, DomainName • ClientId, DomainName , NodeId
ThroughputThrottle	• ClientId, DomainName • ClientId, DomainName , NodeId
TookCurrent	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId

Metric	Dimensions
TookTimeInMillis	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
TookTotal	• ClientId, DomainName • ClientId, DomainName , InstanceId • ClientId, DomainName , NodeId
TotalPointInTime	• ClientId, DomainName , NodeId
TotalRemoteReindexCallsFailed	• ClientId, DomainName , RemoteDomain
TotalRemoteReindexCallsSucceeded	• ClientId, DomainName , RemoteDomain
VolumeStalledIOCheck	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmCPUUtilization	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmConcurrentSearchLatency	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmConcurrentSearchRate	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmFreeStorageSpace	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmJVMGCOldCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
WarmJVMGCOldCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmJVMGCYoungCollectionCount	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmJVMGCYoungCollectionTime	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmJVMMemoryPressure	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmNodes	• ClientId, DomainName
WarmOldGenJVMMemoryPressure	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmSearchLatency	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmSearchRate	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmSearchableDocuments	• ClientId, DomainName
WarmStorageSpaceUtilization	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmSysMemoryUtilization	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmThreadpoolIndexSearcherQueue	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
WarmThreadpoolIndexSearcher Rejected	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmThreadpoolIndexSearcher Threads	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmThreadpoolSearchQueue	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmThreadpoolSearchRejected	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmThreadpoolSearchThreads	• ClientId, DomainName • ClientId, DomainName , NodeId
WarmToColdMigrationFailureCount	• ClientId, DomainName
WarmToColdMigrationLatency	• ClientId, DomainName
WarmToColdMigrationQueueSize	• ClientId, DomainName
WarmToColdMigrationSuccessCount	• ClientId, DomainName
WarmToHotMigrationQueueSize	• ClientId, DomainName
WorkloadCPUCancellations	• ClientId, DomainName
WorkloadCPURejections	• ClientId, DomainName
WorkloadManagementEnabled	• ClientId, DomainName
WorkloadMemoryCancellations	• ClientId, DomainName
WorkloadQueryCompletions	• ClientId, DomainName
WriteIOPS	• ClientId, DomainName • ClientId, DomainName , NodeId

Metric	Dimensions
WriteIOPSMicroBursting	• ClientId, DomainName • ClientId, DomainName , NodeId
WriteLatency	• ClientId, DomainName • ClientId, DomainName , NodeId
WriteThroughput	• ClientId, DomainName • ClientId, DomainName , NodeId
WriteThroughputMicroBursting	• ClientId, DomainName • ClientId, DomainName , NodeId
warmQueryFailure	• ClientId, DomainName • ClientId, DomainName , NodeId
warmQuerySuccess	• ClientId, DomainName • ClientId, DomainName , NodeId

AWS/ElastiCache

The AWS/ElastiCache namespace includes enriched metrics for the following resource types.

- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::ReplicationGroup

AWS::ElastiCache::CacheCluster

Metric	Dimensions
ActiveDefragHits	• CacheClusterId • CacheClusterId , CacheNodeId
AllocatorFragmentationBytes	• CacheClusterId , CacheNodeId

Metric	Dimensions
AllocatorFragmentationRatio	• CacheClusterId ,CacheNodeId
AuthenticationFailures	• CacheClusterId • CacheClusterId ,CacheNodeId
BlockedConnections	• CacheClusterId ,CacheNodeId
BytesReadFromDisk	• CacheClusterId ,CacheNodeId
BytesReadIntoMemcached	• CacheClusterId • CacheClusterId ,CacheNodeId
BytesUsedForCache	• CacheClusterId • CacheClusterId ,CacheNodeId • CacheClusterId ,CacheNodeId ,Tier
BytesUsedForCacheItems	• CacheClusterId • CacheClusterId ,CacheNodeId
BytesUsedForHash	• CacheClusterId • CacheClusterId ,CacheNodeId
BytesWrittenOutFromMemcached	• CacheClusterId • CacheClusterId ,CacheNodeId
BytesWrittenToDisk	• CacheClusterId ,CacheNodeId
CPUCreditBalance	• CacheClusterId • CacheClusterId ,CacheNodeId
CPUCreditUsage	• CacheClusterId • CacheClusterId ,CacheNodeId
CPUUtilization	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
CacheHitRate	• CacheClusterId • CacheClusterId ,CacheNodeId
CacheHits	• CacheClusterId • CacheClusterId ,CacheNodeId
CacheMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
CasBadval	• CacheClusterId • CacheClusterId ,CacheNodeId
CasHits	• CacheClusterId • CacheClusterId ,CacheNodeId
CasMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
ChannelAuthorizationFailures	• CacheClusterId ,CacheNodeId
ClusterBasedCmds	• CacheClusterId ,CacheNodeId
ClusterBasedCmdsLatency	• CacheClusterId ,CacheNodeId
CmdConfigGet	• CacheClusterId • CacheClusterId ,CacheNodeId
CmdConfigSet	• CacheClusterId • CacheClusterId ,CacheNodeId
CmdFlush	• CacheClusterId • CacheClusterId ,CacheNodeId
CmdGet	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
CmdSet	• CacheClusterId • CacheClusterId ,CacheNodeId
CmdTouch	• CacheClusterId • CacheClusterId ,CacheNodeId
CommandAuthorizationFailures	• CacheClusterId • CacheClusterId ,CacheNodeId
CurrConfig	• CacheClusterId • CacheClusterId ,CacheNodeId
CurrConnections	• CacheClusterId • CacheClusterId ,CacheNodeId
CurrItems	• CacheClusterId • CacheClusterId ,CacheNodeId • CacheClusterId ,CacheNodeId ,Tier
CurrItemsWithVolatileFields	• CacheClusterId ,CacheNodeId
CurrVolatileItems	• CacheClusterId ,CacheNodeId
DB0AverageTTL	• CacheClusterId • CacheClusterId ,CacheNodeId
DatabaseAuthorizationFailures	• CacheClusterId ,CacheNodeId
DatabaseCapacityUsageCounte dForEvictPercentage	• CacheClusterId ,CacheNodeId
DatabaseCapacityUsagePercentage	• CacheClusterId ,CacheNodeId
DatabaseMemoryUsageCountedF orEvictPercentage	• CacheClusterId ,CacheNodeId

Metric	Dimensions
DatabaseMemoryUsagePercentage	• CacheClusterId • CacheClusterId ,CacheNodeId
DecrHits	• CacheClusterId • CacheClusterId ,CacheNodeId
DecrMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
DeleteHits	• CacheClusterId • CacheClusterId ,CacheNodeId
DeleteMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
DurabilityBufferExceededErrorCount	• CacheClusterId ,CacheNodeId
DurabilityLag	• CacheClusterId ,CacheNodeId
EngineCPUUtilization	• CacheClusterId • CacheClusterId ,CacheNodeId
ErrorCount	• CacheClusterId ,CacheNodeId
EvalBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
EvalBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
EvictedUnfetched	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
Evictions	• CacheClusterId • CacheClusterId ,CacheNodeId
ExpiredUnfetched	• CacheClusterId • CacheClusterId ,CacheNodeId
FreeableMemory	• CacheClusterId • CacheClusterId ,CacheNodeId
GeoSpatialBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
GeoSpatialBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
GetHits	• CacheClusterId • CacheClusterId ,CacheNodeId
GetMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
GetTypeCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
GetTypeCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
HashBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
HashBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
HyperLogLogBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
HyperLogLogBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
IamAuthenticationExpirations	• CacheClusterId ,CacheNodeId
IamAuthenticationThrottling	• CacheClusterId ,CacheNodeId
IncrHits	• CacheClusterId • CacheClusterId ,CacheNodeId
IncrMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
IsMaster	• CacheClusterId • CacheClusterId ,CacheNodeId
JsonBasedCmds	• CacheClusterId ,CacheNodeId
JsonBasedCmdsLatency	• CacheClusterId ,CacheNodeId
JsonBasedGetCmds	• CacheClusterId ,CacheNodeId
JsonBasedGetCmdsLatency	• CacheClusterId ,CacheNodeId
JsonBasedSetCmds	• CacheClusterId ,CacheNodeId
JsonBasedSetCmdsLatency	• CacheClusterId ,CacheNodeId
KeyAuthorizationFailures	• CacheClusterId • CacheClusterId ,CacheNodeId
KeyBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
KeyBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
KeysTracked	• CacheClusterId • CacheClusterId ,CacheNodeId
ListBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
ListBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
MajorPageFaults	• CacheClusterId ,CacheNodeId
MasterLinkHealthStatus	• CacheClusterId • CacheClusterId ,CacheNodeId
MemoryFragmentationRatio	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkBandwidthInAllowance Exceeded	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkBandwidthOutAllowanc eExceeded	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkBaselineMaxUsageInPe rcentage	• CacheClusterId ,CacheNodeId
NetworkBaselineMaxUsageOutP ercentage	• CacheClusterId ,CacheNodeId
NetworkBaselineUsageInPercentage	• CacheClusterId ,CacheNodeId
NetworkBaselineUsageOutPerc entage	• CacheClusterId ,CacheNodeId
NetworkBytesIn	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
NetworkBytesOut	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkConntrackAllowanceExceeded	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkLinkLocalAllowanceExceeded	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkMaxBytesIn	• CacheClusterId ,CacheNodeId
NetworkMaxBytesOut	• CacheClusterId ,CacheNodeId
NetworkMaxPacketsIn	• CacheClusterId ,CacheNodeId
NetworkMaxPacketsOut	• CacheClusterId ,CacheNodeId
NetworkPacketsIn	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkPacketsOut	• CacheClusterId • CacheClusterId ,CacheNodeId
NetworkPacketsPerSecondAllowanceExceeded	• CacheClusterId • CacheClusterId ,CacheNodeId
NewConnections	• CacheClusterId • CacheClusterId ,CacheNodeId
NewItems	• CacheClusterId • CacheClusterId ,CacheNodeId
NonKeyTypeCmds	• CacheClusterId ,CacheNodeId
NonKeyTypeCmdsLatency	• CacheClusterId ,CacheNodeId

Metric	Dimensions
NumItemsReadFromDisk	• CacheClusterId ,CacheNodeId
NumItemsWrittenToDisk	• CacheClusterId ,CacheNodeId
ProcessedCommands	• CacheClusterId ,CacheNodeId
PubSubBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
PubSubBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
PubSubChannels	• CacheClusterId ,CacheNodeId
PubSubShardChannels	• CacheClusterId ,CacheNodeId
Reclaimed	• CacheClusterId • CacheClusterId ,CacheNodeId
ReclaimedFields	• CacheClusterId ,CacheNodeId
RejectedConnections	• CacheClusterId ,CacheNodeId
ReplicationBytes	• CacheClusterId • CacheClusterId ,CacheNodeId
ReplicationLag	• CacheClusterId • CacheClusterId ,CacheNodeId
SaveInProgress	• CacheClusterId • CacheClusterId ,CacheNodeId
SearchBasedCmds	• CacheClusterId ,CacheNodeId
SearchBasedCmdsLatency	• CacheClusterId ,CacheNodeId
SearchBasedGetCmds	• CacheClusterId ,CacheNodeId

Metric	Dimensions
SearchBasedGetCmdsLatency	• CacheClusterId ,CacheNodeId
SearchBasedSetCmds	• CacheClusterId ,CacheNodeId
SearchBasedSetCmdsLatency	• CacheClusterId ,CacheNodeId
SearchNumberOfIndexes	• CacheClusterId ,CacheNodeId
SearchTotalIndexedDocuments	• CacheClusterId ,CacheNodeId
SearchUsedMemoryBytes	• CacheClusterId ,CacheNodeId
SetBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
SetBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
SetTypeCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
SetTypeCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
SlabsMoved	• CacheClusterId • CacheClusterId ,CacheNodeId
SortedSetBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
SortedSetBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
StreamBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId

Metric	Dimensions
StreamBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
StringBasedCmds	• CacheClusterId • CacheClusterId ,CacheNodeId
StringBasedCmdsLatency	• CacheClusterId • CacheClusterId ,CacheNodeId
SuccessfulReadRequestLatency	• CacheClusterId ,CacheNodeId
SuccessfulWriteRequestLatency	• CacheClusterId ,CacheNodeId
SwapUsage	• CacheClusterId • CacheClusterId ,CacheNodeId
TouchHits	• CacheClusterId • CacheClusterId ,CacheNodeId
TouchMisses	• CacheClusterId • CacheClusterId ,CacheNodeId
TrafficManagementActive	• CacheClusterId ,CacheNodeId
UnusedMemory	• CacheClusterId • CacheClusterId ,CacheNodeId
UsedMemoryDataset	• CacheClusterId ,CacheNodeId

AWS::ElastiCache::ReplicationGroup

Metric	Dimensions
DatabaseCapacityUsageCounte dForEvictPercentage	• NodeGroupId ,ReplicationGroupId • ReplicationGroupId

Metric	Dimensions
DatabaseMemoryUsageCountedForEvictPercentage	• NodeGroupId ,ReplicationGroupId • ReplicationGroupId
EngineCPUUtilization	• NodeGroupId ,ReplicationGroupId • ReplicationGroupId • ReplicationGroupId ,Role

AWS/ElasticBeanstalk

The AWS/ElasticBeanstalk namespace includes enriched metrics for the following resource types.

- AWS::ElasticBeanstalk::Environment

AWS::ElasticBeanstalk::Environment

Metric	Dimensions
ApplicationLatencyP10	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP50	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP75	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP85	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP90	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP95	• EnvironmentName

Metric	Dimensions
	• EnvironmentName ,InstanceId
ApplicationLatencyP99	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationLatencyP99.9	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationRequests2xx	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationRequests3xx	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationRequests4xx	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationRequests5xx	• EnvironmentName • EnvironmentName ,InstanceId
ApplicationRequestsTotal	• EnvironmentName • EnvironmentName ,InstanceId
CPUIdle	• EnvironmentName ,InstanceId
CPUIowait	• EnvironmentName ,InstanceId
CPUIrq	• EnvironmentName ,InstanceId
CPUNice	• EnvironmentName ,InstanceId
CPUPrivileged	• EnvironmentName ,InstanceId
CPUSoftirq	• EnvironmentName ,InstanceId
CPUSystem	• EnvironmentName ,InstanceId

Metric	Dimensions
CPUUser	• EnvironmentName ,InstanceId
EnvironmentHealth	• EnvironmentName
InstanceHealth	• EnvironmentName ,InstanceId
InstancesDegraded	• EnvironmentName
InstancesInfo	• EnvironmentName
InstancesNoData	• EnvironmentName
InstancesOk	• EnvironmentName
InstancesPending	• EnvironmentName
InstancesSevere	• EnvironmentName
InstancesUnknown	• EnvironmentName
InstancesWarning	• EnvironmentName
LoadAverage1min	• EnvironmentName ,InstanceId
LoadAverage5min	• EnvironmentName ,InstanceId
RootFilesystemUtil	• EnvironmentName • EnvironmentName ,InstanceId

AWS/ElasticMapReduce

The AWS/ElasticMapReduce namespace includes enriched metrics for the following resource types.

- AWS::EMR::Cluster

AWS::EMR::Cluster

Metric	Dimensions
AppsCompleted	• JobFlowId
AppsFailed	• JobFlowId
AppsKilled	• JobFlowId
AppsPending	• JobFlowId
AppsRunning	• JobFlowId
AppsSubmitted	• JobFlowId
AutoTerminationIsClusterIdle	• JobFlowId
ContainerAllocated	• JobFlowId
ContainerPending	• JobFlowId
ContainerPendingRatio	• JobFlowId
ContainerReserved	• JobFlowId
CoreNodesPending	• JobFlowId
CoreNodesRequested	• JobFlowId
CoreNodesRunning	• JobFlowId
CoreUnitsRequested	• JobFlowId
CoreUnitsRunning	• JobFlowId
CoreVCPURequested	• JobFlowId
CoreVCPURunning	• JobFlowId
HDFSBytesRead	• JobFlowId

Metric	Dimensions
	• JobFlowId ,JobId
HDFSBytesWritten	• JobFlowId • JobFlowId ,JobId
HDFSUtilization	• JobFlowId
IsIdle	• JobFlowId
JobsFailed	• JobFlowId
JobsRunning	• JobFlowId
LiveDataNodes	• JobFlowId
LiveTaskTrackers	• JobFlowId
MRActiveNodes	• JobFlowId
MRDecommissionedNodes	• JobFlowId
MRLostNodes	• JobFlowId
MRTotalNodes	• JobFlowId
MRUnhealthyNodes	• JobFlowId
MapSlotsOpen	• JobFlowId
MapTasksRemaining	• JobFlowId
MapTasksRunning	• JobFlowId
MissingBlocks	• JobFlowId
ReduceSlotsOpen	• JobFlowId
ReduceTasksRemaining	• JobFlowId
ReduceTasksRunning	• JobFlowId

Metric	Dimensions
RemainingMapTasksPerSlot	• JobFlowId
S3BytesRead	• JobFlowId • JobFlowId ,JobId
S3BytesWritten	• JobFlowId • JobFlowId ,JobId
TaskNodesPending	• JobFlowId
TaskNodesRequested	• JobFlowId
TaskNodesRunning	• JobFlowId
TaskUnitsRequested	• JobFlowId
TaskUnitsRunning	• JobFlowId
TaskVCPURequested	• JobFlowId
TaskVCPURunning	• JobFlowId
TotalLoad	• JobFlowId
TotalNodesRequested	• JobFlowId
TotalNodesRunning	• JobFlowId
TotalNotebookKernels	• JobFlowId
TotalUnitsRequested	• JobFlowId
TotalUnitsRunning	• JobFlowId
TotalVCPURequested	• JobFlowId
TotalVCPURunning	• JobFlowId

AWS/EventBridge/Pipes

The AWS/EventBridge/Pipes namespace includes enriched metrics for the following resource types.

- AWS::Pipes::Pipe

AWS::Pipes::Pipe

Metric	Dimensions
Duration	• PipeName
EnrichmentStageDuration	• PipeName
EnrichmentStageFailed	• PipeName
EventCount	• PipeName
EventSize	• PipeName
ExecutionFailed	• PipeName
ExecutionPartiallyFailed	• PipeName
ExecutionTimeout	• PipeName
TargetStageDuration	• PipeName
TargetStageFailed	• PipeName
TargetStagePartiallyFailed	• PipeName
TargetStageSkipped	• PipeName

AWS/Events

The AWS/Events namespace includes enriched metrics for the following resource types.

- AWS::Events::Rule

AWS::Events::Rule

Metric	Dimensions
DeadLetterInvocations	• EventBusName ,RuleName • RuleName
FailedInvocations	• EventBusName ,RuleName • RuleName
IngestiontoInvocationCompleteLatency	• EventBusName ,RuleName
IngestiontoInvocationStartLatency	• EventBusName ,RuleName
Invocations	• EventBusName ,RuleName • RuleName
InvocationsFailedToBeSentToDlq	• EventBusName ,RuleName • RuleName
InvocationsSentToDlq	• EventBusName ,RuleName • RuleName
ThrottledRules	• EventBusName ,RuleName • RuleName
TriggeredRules	• EventBusName ,RuleName • RuleName

AWS/FSx

The AWS/FSx namespace includes enriched metrics for the following resource types.

- AWS::FSx::FileSystem

AWS::FSx::FileSystem

Metric	Dimensions
AgeOfOldestQueuedMessage	• FileSystemId , Publisher
CPUUtilization	• FileServer , FileSystemId • FileSystemId
CapacityPoolReadBytes	• FileSystemId , VolumeId
CapacityPoolReadOperations	• FileSystemId , VolumeId
CapacityPoolWriteBytes	• FileSystemId , VolumeId
CapacityPoolWriteOperations	• FileSystemId , VolumeId
ClientConnections	• FileSystemId
CompressionRatio	• FileSystemId , VolumeId
DataReadBytes	• FileSystemId • FileSystemId , VolumeId
DataReadOperationTime	• FileSystemId , VolumeId
DataReadOperations	• FileSystemId • FileSystemId , VolumeId
DataWriteBytes	• FileSystemId • FileSystemId , VolumeId
DataWriteLatency	• FileSystemId , StorageTargetId
DataWriteOperationTime	• FileSystemId , VolumeId
DataWriteOperations	• FileSystemId • FileSystemId , VolumeId

Metric	Dimensions
DirectoryCreateOperations	• FileSystemId , JobId, StorageTargetId • FileSystemId , StorageTargetId
DirectoryDeleteOperations	• FileSystemId , JobId, StorageTargetId • FileSystemId , StorageTargetId
DiskIopsExceededCheck	• FileSystemId , StorageTargetId
DiskIopsUtilization	• Aggregate , FileSystemId • FileSystemId , StorageTargetId
DiskReadBytes	• Aggregate , FileSystemId • FileSystemId , StorageTargetId
DiskReadOperations	• Aggregate , FileSystemId • FileSystemId , StorageTargetId
DiskThroughputBalance	• FileSystemId
DiskThroughputExceededCheck	• FileSystemId , StorageTargetId
DiskThroughputUtilization	• FileSystemId
DiskWriteBytes	• Aggregate , FileSystemId • FileSystemId , StorageTargetId
DiskWriteOperations	• Aggregate , FileSystemId • FileSystemId , StorageTargetId
FileCreateOperations	• FileSystemId , JobId, StorageTargetId • FileSystemId , StorageTargetId

Metric	Dimensions
FileDeleteOperations	• FileSystemId , JobId, StorageTargetId • FileSystemId , StorageTargetId
FileOpenOperations	• FileSystemId , JobId, StorageTargetId • FileSystemId , StorageTargetId
FileServerCacheHitRatio	• CacheType , FileSystemId • FileServer , FileSystemId
FileServerDiskIopsBalance	• FileServer , FileSystemId • FileSystemId
FileServerDiskIopsUtilization	• FileServer , FileSystemId • FileSystemId
FileServerDiskThroughputBalance	• FileServer , FileSystemId
FileServerDiskThroughputUtilization	• FileServer , FileSystemId • FileSystemId
FilesCapacity	• FileSystemId , VolumeId
FilesUsed	• FileSystemId , VolumeId
FreeDataStorageCapacity	• FileSystemId • FileSystemId , StorageTargetId
FreeStorageCapacity	• FileSystemId
MemoryUtilization	• FileSystemId
MetadataOperationTime	• FileSystemId , VolumeId

Metric	Dimensions
MetadataOperations	• FileSystemId • FileSystemId , VolumeId
NetworkReceivedBytes	• FileServer , FileSystemId
NetworkSentBytes	• FileServer , FileSystemId
NetworkThroughputUtilization	• FileServer , FileSystemId • FileSystemId
RenameOperations	• FileSystemId , JobId, StorageTargetId
RepositoryRenameOperations	• FileSystemId , Publisher
StatOperations	• FileSystemId , JobId, StorageTargetId
StorageCapacity	• Aggregate , DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier • FileSystemId , VolumeId
StorageCapacityUtilization	• Aggregate , DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier , VolumeId • FileSystemId , StorageTargetId • FileSystemId , VolumeId

Metric	Dimensions
StorageUsed	• Aggregate , DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier , VolumeId • FileSystemId , VolumeId
UsedStorageCapacity	• DataType, FileSystemId , StorageTier • DataType, FileSystemId , StorageTier , VolumeId • FileSystemId , VolumeId

AWS/Firehose

The AWS/Firehose namespace includes enriched metrics for the following resource types.

- AWS::KinesisFirehose::DeliveryStream

AWS::KinesisFirehose::DeliveryStream

Metric	Dimensions
BackupToS3.Bytes	• DeliveryStreamName
BackupToS3.DataFreshness	• DeliveryStreamName
BackupToS3.Records	• DeliveryStreamName
BackupToS3.Success	• DeliveryStreamName
BytesPerSecondLimit	• DeliveryStreamName

Metric	Dimensions
	• DeliveryStreamName ,SourcePartitionId
DataReadFromKinesisStream.Bytes	• DeliveryStreamName
DataReadFromKinesisStream.Records	• DeliveryStreamName
DataReadFromSource.Backpressured	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DataReadFromSource.Bytes	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DataReadFromSource.Records	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DeliveryToAmazonOpenSearchServerless.AuthFailure	• DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Bytes	• DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.DataFreshness	• DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.DeliveryRejected	• DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Records	• DeliveryStreamName
DeliveryToAmazonOpenSearchServerless.Success	• DeliveryStreamName

Metric	Dimensions
DeliveryToAmazonOpenSearchService.AuthFailure	• DeliveryStreamName
DeliveryToAmazonOpenSearchService.Bytes	• DeliveryStreamName
DeliveryToAmazonOpenSearchService.DataFreshness	• DeliveryStreamName
DeliveryToAmazonOpenSearchService.DeliveryRejected	• DeliveryStreamName
DeliveryToAmazonOpenSearchService.Records	• DeliveryStreamName
DeliveryToAmazonOpenSearchService.Success	• DeliveryStreamName
DeliveryToHttpEndpoint.Bytes	• DeliveryStreamName
DeliveryToHttpEndpoint.DataFreshness	• DeliveryStreamName
DeliveryToHttpEndpoint.ProcessedBytes	• DeliveryStreamName
DeliveryToHttpEndpoint.ProcessedRecords	• DeliveryStreamName
DeliveryToHttpEndpoint.Records	• DeliveryStreamName
DeliveryToHttpEndpoint.Success	• DeliveryStreamName
DeliveryToIceberg.Bytes	• DeliveryStreamName ,IcebergTableName
DeliveryToIceberg.DataFreshness	• DeliveryStreamName ,IcebergTableName

Metric	Dimensions
DeliveryToIceberg.Success	• DeliveryStreamName ,IcebergTableName
DeliveryToIceberg.SuccessfulRowCount	• DeliveryStreamName ,IcebergTableName
DeliveryToRedshift.Bytes	• DeliveryStreamName
DeliveryToRedshift.Records	• DeliveryStreamName
DeliveryToRedshift.Success	• DeliveryStreamName
DeliveryToS3.Bytes	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DeliveryToS3.DataFreshness	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DeliveryToS3.Records	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DeliveryToS3.Success	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
DeliveryToSnowflake.Bytes	• DeliveryStreamName
DeliveryToSnowflake.DataCommitLatency	• DeliveryStreamName
DeliveryToSnowflake.DataFreshness	• DeliveryStreamName
DeliveryToSnowflake.Records	• DeliveryStreamName

Metric	Dimensions
DeliveryToSnowflake.Success	• DeliveryStreamName
DeliveryToSplunk.Bytes	• DeliveryStreamName
DeliveryToSplunk.DataAckLatency	• DeliveryStreamName
DeliveryToSplunk.DataFreshness	• DeliveryStreamName
DeliveryToSplunk.Records	• DeliveryStreamName
DeliveryToSplunk.Success	• DeliveryStreamName
DescribeDeliveryStream.Latency	• DeliveryStreamName
DescribeDeliveryStream.Requests	• DeliveryStreamName
ExecuteProcessing.Duration	• DeliveryStreamName
ExecuteProcessing.Success	• DeliveryStreamName
FailedConversion.Bytes	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
FailedConversion.Records	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
FailedValidation.Bytes	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
FailedValidation.Records	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
IncomingBytes	• DeliveryStreamName

Metric	Dimensions
IncomingPutRequests	• DeliveryStreamName
IncomingRecords	• DeliveryStreamName
KMSKeyAccessDenied	• DeliveryStreamName
KMSKeyDisabled	• DeliveryStreamName
KMSKeyInvalidState	• DeliveryStreamName
KMSKeyNotFound	• DeliveryStreamName
KafkaOffsetLag	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
KinesisMillisBehindLatest	• DeliveryStreamName
ListDeliveryStreams.Latency	• DeliveryStreamName
ListDeliveryStreams.Requests	• DeliveryStreamName
OutputDecompressedBytes.Failed	• DeliveryStreamName
OutputDecompressedBytes.Success	• DeliveryStreamName
OutputDecompressedRecords.Failed	• DeliveryStreamName
OutputDecompressedRecords.Success	• DeliveryStreamName
PutRecord.Bytes	• DeliveryStreamName
PutRecord.Latency	• DeliveryStreamName
PutRecord.Requests	• DeliveryStreamName
PutRecordBatch.Bytes	• DeliveryStreamName

Metric	Dimensions
PutRecordBatch.Latency	• DeliveryStreamName
PutRecordBatch.Records	• DeliveryStreamName
PutRecordBatch.Requests	• DeliveryStreamName
PutRequestsPerSecondLimit	• DeliveryStreamName
RecordsPerSecondLimit	• DeliveryStreamName
SourceThrottled.Delay	• DeliveryStreamName
SourceToDelivery.DataFreshness	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
SucceedConversion.Bytes	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
SucceedConversion.Records	• DeliveryStreamName • DeliveryStreamName ,SourcePartitionId
SucceedProcessing.Bytes	• DeliveryStreamName
SucceedProcessing.Records	• DeliveryStreamName
ThrottledDescribeStream	• DeliveryStreamName
ThrottledGetRecords	• DeliveryStreamName
ThrottledGetShardIterator	• DeliveryStreamName
ThrottledRecords	• DeliveryStreamName
UpdateDeliveryStream.Latency	• DeliveryStreamName

Metric	Dimensions
UpdateDeliveryStream.Requests	• DeliveryStreamName

AWS/FraudDetector

The AWS/FraudDetector namespace includes enriched metrics for the following resource types.

- AWS::FraudDetector::Detector

AWS::FraudDetector::Detector

Metric	Dimensions
GetEventPrediction	• DetectorID
GetEventPrediction4xxError	• DetectorID
GetEventPrediction5xxError	• DetectorID
GetEventPredictionLatency	• DetectorID
ModelInvocation	• DetectorID ,DetectorVersionID , ModelEndpoint • DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
ModelInvocationError	• DetectorID ,DetectorVersionID , ModelEndpoint • DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
ModelInvocationLatency	• DetectorID ,DetectorVersionID , ModelEndpoint

Metric	Dimensions
	• DetectorID ,DetectorVersionID , ModelID, ModelType ,ModelVersionNumber
OutcomeReturned	• DetectorID ,DetectorVersionID , OutcomeName • DetectorID ,OutcomeName
Prediction	• DetectorID ,DetectorVersionID
PredictionError	• DetectorID ,DetectorVersionID
PredictionLatency	• DetectorID ,DetectorVersionID
RuleEvaluateFalse	• DetectorID ,DetectorVersionID , RuleID
RuleEvaluateTrue	• DetectorID ,DetectorVersionID , RuleID
RuleNotEvaluated	• DetectorID ,DetectorVersionID , RuleID
VariableUsed	• DetectorID ,DetectorVersionID , VariableName

AWS/GameLift

The AWS/GameLift namespace includes enriched metrics for the following resource types.

- AWS::GameLift::GameSessionQueue
- AWS::GameLift::MatchmakingConfiguration

AWS::GameLift::GameSessionQueue

Metric	Dimensions
AverageWaitTime	• Location, QueueName • QueueName
FirstChoiceNotViable	• QueueName
FirstChoiceOutOfCapacity	• QueueName
GameSessionPlaced	• LocationName , QueueName
LowestLatencyPlacement	• QueueName
LowestPricePlacement	• QueueName
PlacementApNortheast1	• QueueName
PlacementApNortheast2	• QueueName
PlacementApSouth1	• QueueName
PlacementApSoutheast2	• QueueName
PlacementEuCentral1	• QueueName
PlacementEuNorth1	• QueueName
PlacementEuWest1	• QueueName
PlacementEuWest2	• QueueName
PlacementMeSouth1	• QueueName
PlacementSaEast1	• QueueName
PlacementUsEast1	• QueueName
PlacementUsEast2	• QueueName

Metric	Dimensions
PlacementUsWest1	• QueueName
PlacementUsWest2	• QueueName
PlacementsCanceled	• QueueName
PlacementsStarted	• QueueName
PlacementsSucceeded	• QueueName
PlacementsTimedOut	• QueueName
QueueDepth	• Location, QueueName • QueueName

AWS::GameLift::MatchmakingConfiguration

Metric	Dimensions
CurrentTickets	• ConfigurationName
MatchesAccepted	• ConfigurationName
MatchesCreated	• ConfigurationName
MatchesPlaced	• ConfigurationName
MatchesRejected	• ConfigurationName
MatchmakingSearchTime	• ConfigurationName
PlayersStarted	• ConfigurationName
TicketsStarted	• ConfigurationName
TicketsTimedOut	• ConfigurationName
TimeToMatch	• ConfigurationName

Metric	Dimensions
TimeToTicketCancel	• ConfigurationName
TimeToTicketSuccess	• ConfigurationName

AWS/GatewayELB

The AWS/GatewayELB namespace includes enriched metrics for the following resource types.

- AWS::ElasticLoadBalancingV2::LoadBalancer

AWS::ElasticLoadBalancingV2::LoadBalancer

Metric	Dimensions
ActiveFlowCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ConsumedLCUs	• LoadBalancer
HealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
NewFlowCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ProcessedBytes	• AvailabilityZone ,LoadBalancer • LoadBalancer
UnHealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup

AWS/IVSChat

The AWS/IVSChat namespace includes enriched metrics for the following resource types.

- AWS::IVSChat::LoggingConfiguration

AWS::IVSChat::LoggingConfiguration

Metric	Dimensions
LogDestinationAccessDeniedError	• LoggingConfiguration
LogDestinationErrors	• LoggingConfiguration
LogDestinationResourceNotFoundErrors	• LoggingConfiguration

AWS/IoT

The AWS/IoT namespace includes enriched metrics for the following resource types.

- AWS::IoT::CACertificate
- AWS::IoT::ScheduledAudit
- AWS::IoT::SecurityProfile
- AWS::IoT::TopicRule

AWS::IoT::CACertificate

Metric	Dimensions
ProvisionThing.ClientError	• CaCertificateId
ProvisionThing.ServerError	• CaCertificateId
ProvisionThing.Success	• CaCertificateId

AWS::IoT::ScheduledAudit

Metric	Dimensions
MisconfiguredDeviceDefender Notification	• CheckName , ScheduledAuditName
NonCompliantResources	• CheckName , ScheduledAuditName
ResourcesEvaluated	• CheckName , ScheduledAuditName

AWS::IoT::SecurityProfile

Metric	Dimensions
BehaviorEvaluationCompleted	• BehaviorName , SecurityProfileNam e
BehaviorEvaluationSkipped	• BehaviorName , SecurityProfileNam e
MisconfiguredDeviceDefender Notification	• BehaviorName , SecurityProfileNam e
NumOfMetricsExceedingSizeLimit	• BehaviorName , SecurityProfileNam e
NumOfMetricsExported	• BehaviorName , SecurityProfileNam e
NumOfMetricsSkipped	• BehaviorName , SecurityProfileNam e
Violations	• BehaviorName , SecurityProfileNam e • SecurityProfileName

Metric	Dimensions
ViolationsCleared	• BehaviorName ,SecurityProfileName • SecurityProfileName
ViolationsInvalidated	• BehaviorName ,SecurityProfileName • SecurityProfileName

AWS::IoT::TopicRule

Metric	Dimensions
ErrorActionFailure	• ActionType ,RuleName
ErrorActionHttpDestinationNotEnabled	• ActionType ,RuleName
ErrorActionSuccess	• ActionType ,RuleName
Failure	• ActionType ,RuleName
HttpCode_4XX	• ActionType ,RuleName
HttpCode_5XX	• ActionType ,RuleName
HttpCode_Other	• ActionType ,RuleName
HttpInvalidUrl	• ActionType ,RuleName
HttpRequestTimeout	• ActionType ,RuleName
HttpUnknownHost	• ActionType ,RuleName
ParseError	• RuleName
RequestTimeout	• ActionType ,RuleName

Metric	Dimensions
RuleMessageThrottled	• RuleName
RuleNotFound	• RuleName
RulesExecuted	• RuleName
SaslAuthFailed	• ActionType ,RuleName
Success	• ActionType ,RuleName
TopicMatch	• RuleName

AWS/KMS

The AWS/KMS namespace includes enriched metrics for the following resource types.

- AWS::KMS::Key

AWS::KMS::Key

Metric	Dimensions
SecondsUntilKeyMaterialExpiration	• KeyId

AWS/Kendra

The AWS/Kendra namespace includes enriched metrics for the following resource types.

- AWS::Kendra::DataSource
- AWS::Kendra::Index

AWS::Kendra::DataSource

Metric	Dimensions
DataSourceDocumentCount	• DataSourceId ,IndexId
DataSourceSyncRuntime	• DataSourceId ,IndexId
DocumentsCrawled	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId • DataSourceId ,IndexId
DocumentsFailedToIndex	• DataSourceId ,IndexId
DocumentsIndexed	• DataSourceId ,IndexId
DocumentsSkippedInvalidMetadata	• DataSourceId ,IndexId
DocumentsSkippedNoChange	• DataSourceId ,IndexId
DocumentsSubmittedForDeletion	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId • DataSourceId ,IndexId
DocumentsSubmittedForDeletionFailed	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId • DataSourceId ,IndexId
DocumentsSubmittedForIndexing	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId • DataSourceId ,IndexId
DocumentsSubmittedForIndexingFailed	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId • DataSourceId ,IndexId
DocumentsWithoutAcl	• AwsAccountId ,DataSourceId , DataSourceType ,IndexId

Metric	Dimensions
MetadataFilesCrawled	• DataSourceId , IndexId
PrincipalGroupsScanned	• DataSourceId , IndexId

AWS::Kendra::Index

Metric	Dimensions
ClickFeedbackSubmittedCount	• IndexId
DocumentsFailedToIndex	• IndexId
DocumentsIndexed	• IndexId
GroupProcessed	• IndexId
GroupProcessedButNotPersistent	• IndexId
GroupReceived	• IndexId
GroupUpdateLatencyInMilliSeconds	• IndexId
GroupWithOlderOrderingIdSkipped	• IndexId
IndexDocumentCount	• IndexId
IndexDocumentStorageSize	• IndexId
IndexQueryCount	• IndexId
ProvisionedIndexDocumentCount	• IndexId
ProvisionedIndexStorageSize	• IndexId
RelevantFeedbackSubmittedCount	• IndexId

AWS/Kinesis

The AWS/Kinesis namespace includes enriched metrics for the following resource types.

- AWS::Kinesis::Stream

AWS::Kinesis::Stream

Metric	Dimensions
GetRecords.Bytes	• StreamName
GetRecords.IteratorAgeMilliseconds	• StreamName
GetRecords.Latency	• StreamName
GetRecords.Records	• StreamName
GetRecords.Success	• StreamName
IncomingBytes	• ShardId, StreamName • StreamName
IncomingRecords	• ShardId, StreamName • StreamName
IteratorAgeMilliseconds	• ShardId, StreamName
OutgoingBytes	• ShardId, StreamName
OutgoingRecords	• ShardId, StreamName
PutRecord.Success	• StreamName
PutRecords.Bytes	• StreamName
PutRecords.Success	• StreamName

Metric	Dimensions
ReadProvisionedThroughputExceeded	• ShardId, StreamName • StreamName
SubscribeToShard.RateExceeded	• ConsumerName , StreamName
SubscribeToShard.Success	• ConsumerName , StreamName
SubscribeToShardEvent.Bytes	• ConsumerName , StreamName
SubscribeToShardEvent.MillisBehindLatest	• ConsumerName , StreamName
SubscribeToShardEvent.Records	• ConsumerName , StreamName
SubscribeToShardEvent.Success	• ConsumerName , StreamName
WriteProvisionedThroughputExceeded	• ShardId, StreamName • StreamName

AWS/KinesisAnalytics

The AWS/KinesisAnalytics namespace includes enriched metrics for the following resource types.

- AWS::KinesisAnalyticsV2::Application

AWS::KinesisAnalyticsV2::Application

Metric	Dimensions
KPUs	• Application
KPUs-Interactive	• Application
backPressuredTimeMsPerSecond	• Application • Application , Task

Metric	Dimensions
	• Application , Task, TaskParal lelismIndex
busyTimeMsPerSecond	• Application • Application , Task • Application , Task, TaskParal lelismIndex
bytesRequestedPerFetch	• Application , Flow, Id • Application , Flow, Id, ShardId
committedOffsets	• Application , Flow, Topic
containerCPUUtilization	• Application
containerDiskUtilization	• Application
containerMemoryUtilization	• Application
cpuUtilization	• Application
currentInputWatermark	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex • Application , Task, TaskParal lelismIndex
currentOffsets	• Application , Flow, Topic

Metric	Dimensions
currentOutputWatermark	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex
downtime	• Application
fullRestarts	• Application
heapMemoryUtilization	• Application
idleTimeMsPerSecond	• Application • Application , Task • Application , Task, TaskParal lelismIndex
jobmanagerFileDescriptorsMax	• Application
jobmanagerFileDescriptorsOpen	• Application
jobmanagerHeapMemoryUtilization	• Application
jobmanagerMetaspaceMemoryUt ilization	• Application
lastCheckpointDuration	• Application
lastCheckpointSize	• Application
managedMemoryTotal	• Application
managedMemoryUsed	• Application
managedMemoryUtilization	• Application
metaspaceMemoryUtilization	• Application

Metric	Dimensions
millisBehindLatest	• Application , Flow, Id • Application , Flow, Id, ShardId
numLateRecordsDropped	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex
numRecordsIn	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex • Application , Task, TaskParal lelismIndex
numRecordsInPerSecond	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex • Application , Task, TaskParal lelismIndex
numRecordsOut	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex • Application , Task, TaskParal lelismIndex

Metric	Dimensions
numRecordsOutPerSecond	• Application • Application , Task • Application , Task, TaskOperator • Application , Task, TaskOperator , TaskOperatorParallelismIndex • Application , Task, TaskParal lelismIndex
numRestarts	• Application
numberOfFailedCheckpoints	• Application
oldGenerationGCCount	• Application
oldGenerationGCTime	• Application
taskmanagerFileDescriptorsMax	• Application
taskmanagerFileDescriptorsOpen	• Application
threadCount	• Application
threadsCount	• Application
uptime	• Application
zeppelinCpuUtilization	• Application
zeppelinHeapMemoryUtilization	• Application
zeppelinServerUptime	• Application
zeppelinThreadCount	• Application
zeppelinWaitingJobs	• Application

AWS/Lambda

The AWS/Lambda namespace includes enriched metrics for the following resource types.

- AWS::Lambda::EventSourceMapping
- AWS::Lambda::Function

AWS::Lambda::EventSourceMapping

Metric	Dimensions
CommittedEventCount	• EventSourceMappingUUID
DeletedEventCount	• EventSourceMappingUUID
DroppedEventCount	• EventSourceMappingUUID
EventPollerThroughputInBytes	• EventSourceMappingUUID
EventPollerUnit	• EventSourceMappingUUID
FailedInvokeEventCount	• EventSourceMappingUUID
FilteredOutEventCount	• EventSourceMappingUUID
InvokedEventCount	• EventSourceMappingUUID
IteratorAge	• EventSourceMappingUUID
MaxOffsetLag	• EventSourceMappingUUID
OnFailureDestinationDeliveredEventCount	• EventSourceMappingUUID
PolledEventCount	• EventSourceMappingUUID
ProvisionedPollers	• EventSourceMappingUUID
SumOffsetLag	• EventSourceMappingUUID

AWS::Lambda::Function

Metric	Dimensions
AsyncEventAge	• FunctionName • FunctionName ,Resource
AsyncEventsDropped	• FunctionName • FunctionName ,Resource
AsyncEventsReceived	• FunctionName • FunctionName ,Resource
CPUThrottles	• FunctionName • FunctionName ,Resource
ConcurrencyThrottles	• FunctionName • FunctionName ,Resource
ConcurrentExecutions	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DeadLetterErrors	• FunctionName • FunctionName ,Resource
DestinationDeliveryFailures	• FunctionName • FunctionName ,Resource
DiskThrottles	• FunctionName • FunctionName ,Resource
DurableExecutionDuration	• ExecutedVersion ,FunctionName , Resource • FunctionName

Metric	Dimensions
	• FunctionName ,Resource
DurableExecutionFailed	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DurableExecutionOperations	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DurableExecutionStarted	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DurableExecutionStopped	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DurableExecutionStorageWrit tenBytes	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
DurableExecutionSucceeded	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource

Metric	Dimensions
DurableExecutionTimedOut	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Duration	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Errors	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Invocations	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
IteratorAge	• FunctionName • FunctionName ,Resource
MemoryThrottles	• FunctionName • FunctionName ,Resource
OffsetLag	• FunctionName • FunctionName ,Resource
PostRuntimeExtensionsDuration	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource

Metric	Dimensions
ProvisionedConcurrencyInvocations	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
ProvisionedConcurrencySpilloverInvocations	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
ProvisionedConcurrencyUtilization	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
ProvisionedConcurrentExecutions	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
RecursiveInvocationsDropped	• FunctionName • FunctionName ,Resource
SignatureValidationErrors	• FunctionName • FunctionName ,Resource
StreamedOutboundBytes	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource

Metric	Dimensions
StreamedOutboundThroughput	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Throttles	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
TimeToFirstByteLatency	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
TimeToLastByteLatency	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Url4xxCount	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource
Url5xxCount	• ExecutedVersion ,FunctionName , Resource • FunctionName • FunctionName ,Resource

Metric	Dimensions
UrlRequestCount	• ExecutedVersion , FunctionName , Resource • FunctionName • FunctionName , Resource
UrlRequestLatency	• ExecutedVersion , FunctionName , Resource • FunctionName • FunctionName , Resource

AWS/MWAA

The AWS/MWAA namespace includes enriched metrics for the following resource types.

- AWS::MWAA::Environment

AWS::MWAA::Environment

Metric	Dimensions
ActiveConnectionCount	• Environment
ApproximateAgeOfOldestTask	• Environment
CPUUtilization	• Cluster, Environment • DatabaseRole , Environment
DatabaseConnections	• DatabaseRole , Environment
DiskQueueDepth	• DatabaseRole , Environment
EphemeralStorageReserved	• Cluster, Environment
EphemeralStorageUtilized	• Cluster, Environment

Metric	Dimensions
FreeableMemory	• DatabaseRole , Environment
MemoryUtilization	• Cluster, Environment
NetworkReceiveThroughput	• DatabaseRole , Environment
NetworkRxBytes	• Cluster, Environment
NetworkTransmitThroughput	• DatabaseRole , Environment
NetworkTxBytes	• Cluster, Environment
QueuedTasks	• Environment
ReadIOPS	• DatabaseRole , Environment
ReadLatency	• DatabaseRole , Environment
ReadThroughput	• DatabaseRole , Environment
RunningTasks	• Environment
WriteIOPS	• DatabaseRole , Environment
WriteLatency	• DatabaseRole , Environment
WriteThroughput	• DatabaseRole , Environment

AWS/MediaTailor

The AWS/MediaTailor namespace includes enriched metrics for the following resource types.

- AWS::MediaTailor::Channel

AWS::MediaTailor::Channel

Metric	Dimensions
4xxErrorCount	• Channel • Channel, Output
5xxErrorCount	• Channel • Channel, Output
RequestCount	• Channel • Channel, Output
TotalTime	• Channel • Channel, Output

AWS/MemoryDB

The AWS/MemoryDB namespace includes enriched metrics for the following resource types.

- AWS::MemoryDB::Cluster

AWS::MemoryDB::Cluster

Metric	Dimensions
ActiveDefragHits	• ClusterName • ClusterName , NodeName
AuthenticationFailures	• ClusterName • ClusterName , NodeName
BytesReadFromDisk	• ClusterName , NodeName
BytesUsedForMemoryDB	• ClusterName • ClusterName , NodeName

Metric	Dimensions
BytesWrittenToDisk	• ClusterName ,NodeName
CPUUtilization	• ClusterName • ClusterName ,NodeName
ChannelAuthorizationFailures	• ClusterName • ClusterName ,NodeName
CommandAuthorizationFailures	• ClusterName • ClusterName ,NodeName
CurrConnections	• ClusterName • ClusterName ,NodeName
CurrItems	• ClusterName • ClusterName ,NodeName
DB0AverageTTL	• ClusterName • ClusterName ,NodeName
DatabaseCapacityUsagePercentage	• ClusterName • ClusterName ,NodeName
DatabaseMemoryUsagePercentage	• ClusterName • ClusterName ,NodeName
EngineCPUUtilization	• ClusterName • ClusterName ,NodeName
ErrorCount	• ClusterName • ClusterName ,NodeName
EvalBasedCmds	• ClusterName • ClusterName ,NodeName

Metric	Dimensions
Evictions	• ClusterName • ClusterName ,NodeName
FreeableMemory	• ClusterName • ClusterName ,NodeName
GetTypeCmds	• ClusterName • ClusterName ,NodeName
HashBasedCmds	• ClusterName • ClusterName ,NodeName
IamAuthenticationExpirations	• ClusterName • ClusterName ,NodeName
IamAuthenticationThrottling	• ClusterName • ClusterName ,NodeName
IsPrimary	• ClusterName • ClusterName ,NodeName
JsonBasedCmds	• ClusterName • ClusterName ,NodeName
JsonBasedGetCmds	• ClusterName • ClusterName ,NodeName
JsonBasedSetCmds	• ClusterName • ClusterName ,NodeName
KeyAuthorizationFailures	• ClusterName • ClusterName ,NodeName
KeyBasedCmds	• ClusterName • ClusterName ,NodeName

Metric	Dimensions
KeysTracked	• ClusterName • ClusterName ,NodeName
KeyspaceHits	• ClusterName • ClusterName ,NodeName
KeyspaceMisses	• ClusterName • ClusterName ,NodeName
ListBasedCmds	• ClusterName • ClusterName ,NodeName
MaxReplicationThroughput	• ClusterName • ClusterName ,NodeName
MemoryFragmentationRatio	• ClusterName • ClusterName ,NodeName
NetworkBandwidthInAllowance Exceeded	• ClusterName • ClusterName ,NodeName
NetworkBandwidthOutAllowanc eExceeded	• ClusterName • ClusterName ,NodeName
NetworkBytesIn	• ClusterName • ClusterName ,NodeName
NetworkBytesOut	• ClusterName • ClusterName ,NodeName
NetworkConntrackAllowanceEx ceeded	• ClusterName • ClusterName ,NodeName
NetworkMaxBytesIn	• ClusterName • ClusterName ,NodeName

Metric	Dimensions
NetworkMaxBytesOut	• ClusterName • ClusterName ,NodeName
NetworkMaxPacketsIn	• ClusterName • ClusterName ,NodeName
NetworkMaxPacketsOut	• ClusterName • ClusterName ,NodeName
NetworkPacketsIn	• ClusterName • ClusterName ,NodeName
NetworkPacketsOut	• ClusterName • ClusterName ,NodeName
NetworkPacketsPerSecondAllowanceExceeded	• ClusterName • ClusterName ,NodeName
NewConnections	• ClusterName • ClusterName ,NodeName
NonKeyTypeCmds	• ClusterName • ClusterName ,NodeName
NumItemsReadFromDisk	• ClusterName • ClusterName ,NodeName
NumItemsWrittenToDisk	• ClusterName • ClusterName ,NodeName
PrimaryLinkHealthStatus	• ClusterName • ClusterName ,NodeName
PubSubBasedCmds	• ClusterName • ClusterName ,NodeName

Metric	Dimensions
Reclaimed	• ClusterName • ClusterName ,NodeName
ReplicationBytes	• ClusterName • ClusterName ,NodeName
ReplicationDelayedWriteCommands	• ClusterName • ClusterName ,NodeName
ReplicationLag	• ClusterName • ClusterName ,NodeName
SearchBasedCmds	• ClusterName • ClusterName ,NodeName
SearchBasedGetCmds	• ClusterName • ClusterName ,NodeName
SearchBasedSetCmds	• ClusterName • ClusterName ,NodeName
SearchNumberOfIndexedKeys	• ClusterName • ClusterName ,NodeName
SearchNumberOfIndexes	• ClusterName • ClusterName ,NodeName
SearchTotalIndexSize	• ClusterName • ClusterName ,NodeName
SetBasedCmds	• ClusterName • ClusterName ,NodeName
SetTypeCmds	• ClusterName • ClusterName ,NodeName

Metric	Dimensions
SortedSetBasedCmds	• ClusterName • ClusterName ,NodeName
StreamBasedCmds	• ClusterName • ClusterName ,NodeName
StringBasedCmds	• ClusterName • ClusterName ,NodeName
SuccessfulReadRequestLatency	• ClusterName • ClusterName ,NodeName
SuccessfulWriteRequestLatency	• ClusterName • ClusterName ,NodeName
SwapUsage	• ClusterName • ClusterName ,NodeName

AWS/NATGateway

The AWS/NATGateway namespace includes enriched metrics for the following resource types.

- AWS::EC2::NatGateway

AWS::EC2::NatGateway

Metric	Dimensions
ActiveConnectionCount	• AvailabilityZone ,NatGatewayId • NatGatewayId
BytesInFromDestination	• AvailabilityZone ,NatGatewayId • NatGatewayId
BytesInFromSource	• AvailabilityZone ,NatGatewayId

Metric	Dimensions
	• NatGatewayId
BytesOutToDestination	• AvailabilityZone ,NatGatewayId • NatGatewayId
BytesOutToSource	• AvailabilityZone ,NatGatewayId • NatGatewayId
ConnectionAttemptCount	• AvailabilityZone ,NatGatewayId • NatGatewayId
ConnectionEstablishedCount	• AvailabilityZone ,NatGatewayId • NatGatewayId
ErrorPortAllocation	• AvailabilityZone ,NatGatewayId • NatGatewayId
IdleTimeoutCount	• AvailabilityZone ,NatGatewayId • NatGatewayId
PacketsDropCount	• AvailabilityZone ,NatGatewayId • NatGatewayId
PacketsInFromDestination	• AvailabilityZone ,NatGatewayId • NatGatewayId
PacketsInFromSource	• AvailabilityZone ,NatGatewayId • NatGatewayId
PacketsOutToDestination	• AvailabilityZone ,NatGatewayId • NatGatewayId
PacketsOutToSource	• AvailabilityZone ,NatGatewayId • NatGatewayId

Metric	Dimensions
PeakBytesPerSecond	• AvailabilityZone ,NatGatewayId • NatGatewayId
PeakPacketsPerSecond	• AvailabilityZone ,NatGatewayId • NatGatewayId

AWS/NetworkELB

The AWS/NetworkELB namespace includes enriched metrics for the following resource types.

- AWS::ElasticLoadBalancingV2::LoadBalancer

AWS::ElasticLoadBalancingV2::LoadBalancer

Metric	Dimensions
ActiveFlowCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer ,TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
ActiveFlowCount_TCP	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer ,TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
ActiveFlowCount_TLS	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer ,TargetGroup • LoadBalancer

Metric	Dimensions
ActiveFlowCount_UDP	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
ActiveZonalShiftHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
ClientTLSNegotiationErrorCount	• LoadBalancer
ConsumedLCUs	• LoadBalancer
ConsumedLCUs_TCP	• LoadBalancer
ConsumedLCUs_TLS	• LoadBalancer
ConsumedLCUs_UDP	• LoadBalancer
HealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup
NewFlowCount	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup

Metric	Dimensions
NewFlowCount_TCP	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
NewFlowCount_TLS	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer
NewFlowCount_UDP	• AvailabilityZone ,LoadBalancer • AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer • LoadBalancer ,TargetGroup
PeakPacketsPerSecond	• AvailabilityZone ,LoadBalancer • LoadBalancer
PortAllocationErrorCount	• AvailabilityZone ,LoadBalancer • LoadBalancer
ProcessedBytes	• AvailabilityZone ,LoadBalancer • LoadBalancer
ProcessedBytes_TCP	• AvailabilityZone ,LoadBalancer • LoadBalancer
ProcessedBytes_TLS	• AvailabilityZone ,LoadBalancer • LoadBalancer

Metric	Dimensions
ProcessedBytes_UDP	• AvailabilityZone ,LoadBalancer • LoadBalancer
ProcessedPackets	• AvailabilityZone ,LoadBalancer • LoadBalancer
TCP_Client_Reset_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
TCP_ELB_Reset_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
TCP_Target_Reset_Count	• AvailabilityZone ,LoadBalancer • LoadBalancer
TargetTLSNegotiationErrorCount	• LoadBalancer
UnHealthyHostCount	• AvailabilityZone ,LoadBalancer , TargetGroup • LoadBalancer ,TargetGroup

AWS/NetworkFirewall

The AWS/NetworkFirewall namespace includes enriched metrics for the following resource types.

- AWS::NetworkFirewall::Firewall

AWS::NetworkFirewall::Firewall

Metric	Dimensions
DroppedPackets	• AvailabilityZone ,Engine, FirewallName

Metric	Dimensions
InvalidDroppedPackets	• AvailabilityZone , Engine, FirewallName
OtherDroppedPackets	• AvailabilityZone , Engine, FirewallName
Packets	• AvailabilityZone , CustomAction , Engine, FirewallName
PassedPackets	• AvailabilityZone , Engine, FirewallName
ReceivedPackets	• AvailabilityZone , Engine, FirewallName
RejectedPackets	• AvailabilityZone , Engine, FirewallName
StreamExceptionPolicyPackets	• AvailabilityZone , Engine, FirewallName
TLSDroppedPackets	• AvailabilityZone , Engine, FirewallName
TLSErrors	• AvailabilityZone , Engine, FirewallName
TLSPassedPackets	• AvailabilityZone , Engine, FirewallName
TLSReceivedPackets	• AvailabilityZone , Engine, FirewallName
TLSRejectedPackets	• AvailabilityZone , Engine, FirewallName

Metric	Dimensions
TLSRevocationStatusOKConnections	• AvailabilityZone ,Engine, FirewallName
TLSRevocationStatusRevokedConnections	• AvailabilityZone ,Engine, FirewallName
TLSRevocationStatusUnknownConnections	• AvailabilityZone ,Engine, FirewallName
TLSTimedOutConnections	• AvailabilityZone ,Engine, FirewallName

AWS/Prometheus

The AWS/Prometheus namespace includes enriched metrics for the following resource types.

- AWS::APS::RuleGroupsNamespace
- AWS::APS::Workspace

AWS::APS::RuleGroupsNamespace

Metric	Dimensions
RuleEvaluationFailures	• RuleGroup ,Workspace
RuleEvaluations	• RuleGroup ,Workspace
RuleGroupIterationsMissed	• RuleGroup ,Workspace
RuleGroupLastEvaluationDuration	• RuleGroup ,Workspace

AWS::APS::Workspace

Metric	Dimensions
ActiveSeriesLimitPerLabelSet	• LabelSet, Workspace
ActiveSeriesPerLabelSet	• LabelSet, Workspace
AlertManagerAlertsReceived	• Workspace
AlertManagerNotificationsFailed	• Workspace
AlertManagerNotificationsFailedByIntegration	• Workspace
AlertManagerNotificationsThrottled	• Workspace
AlertManagerNotificationsThrottledByIntegration	• Workspace
DiscardedSamples	• Reason, Workspace
DiscardedSamplesPerLabelSet	• LabelSet, Reason, Workspace
DiscardedSeries	• Reason, Workspace
DiscardedSeriesPerLabelSet	• LabelSet, Reason, Workspace
IngestionRatePerLabelSet	• LabelSet, Workspace
QuerySamplesProcessed	• Workspace
SecretFetchFailure	• Workspace

AWS/RDS

The AWS/RDS namespace includes enriched metrics for the following resource types.

- AWS::RDS::DBCluster

- AWS::RDS::DBInstance

AWS::RDS::DBCluster

Metric	Dimensions
AbortedClients	• DBClusterIdentifier ,Role
ActiveTransactions	• DBClusterIdentifier ,Role
AuroraBinlogReplicaLag	• DBClusterIdentifier ,Role
AuroraDMLRejectedMasterFull	• DBClusterIdentifier ,Role
AuroraReplicaLagMaximum	• DBClusterIdentifier ,Role
AuroraReplicaLagMinimum	• DBClusterIdentifier ,Role
AuroraVolumeBytesLeftTotal	• DBClusterIdentifier ,Role
Aurora_pq_request_attempted	• DBClusterIdentifier ,Role
Aurora_pq_request_executed	• DBClusterIdentifier ,Role
Aurora_pq_request_failed	• DBClusterIdentifier ,Role
Aurora_pq_request_in_progress	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chose n_below_min_rows	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chose n_column_bit	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chose n_column_geometry	• DBClusterIdentifier ,Role

Metric	Dimensions
Aurora_pq_request_not_chosen_column_lob	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_column_virtual	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_custom_charset	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_fast_ddl	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_full_text_index	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_high_buffer_pool_pct	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_index_hint	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_innodb_table_format	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_long_trx	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_no_where_clause	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_range_scan	• DBClusterIdentifier ,Role

Metric	Dimensions
Aurora_pq_request_not_chosen_row_length_too_long	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_small_table	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_temporary_table	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_tx_isolation	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_unsupported_access	• DBClusterIdentifier ,Role
Aurora_pq_request_not_chosen_update_delete_stmts	• DBClusterIdentifier ,Role
Aurora_pq_request_throttled	• DBClusterIdentifier ,Role
BlockedTransactions	• DBClusterIdentifier ,Role
BufferCacheHitRatio	• DBClusterIdentifier ,Role
CPUUtilization	• DBClusterIdentifier • DBClusterIdentifier ,Role
CommitLatency	• DBClusterIdentifier ,Role
CommitThroughput	• DBClusterIdentifier ,Role
ConnectionAttempts	• DBClusterIdentifier ,Role
DDLLatency	• DBClusterIdentifier ,Role
DDLThroughput	• DBClusterIdentifier ,Role
DMLLatency	• DBClusterIdentifier ,Role

Metric	Dimensions
DMLThroughput	• DBClusterIdentifier ,Role
DatabaseConnections	• DBClusterIdentifier • DBClusterIdentifier ,Role
Deadlocks	• DBClusterIdentifier ,Role
DeleteLatency	• DBClusterIdentifier ,Role
DeleteThroughput	• DBClusterIdentifier ,Role
DiskQueueDepth	• DBClusterIdentifier ,Role
EBSByteBalance%	• DBClusterIdentifier ,Role
EBSIOBalance%	• DBClusterIdentifier ,Role
EngineUptime	• DBClusterIdentifier ,Role
ForwardingMasterDMLLatency	• DBClusterIdentifier ,Role
ForwardingMasterDMLThroughput	• DBClusterIdentifier ,Role
ForwardingMasterOpenSessions	• DBClusterIdentifier ,Role
ForwardingReplicaDMLLatency	• DBClusterIdentifier ,Role
ForwardingReplicaDMLThroughput	• DBClusterIdentifier ,Role
ForwardingReplicaOpenSessions	• DBClusterIdentifier ,Role
ForwardingReplicaReadWaitLatency	• DBClusterIdentifier ,Role
ForwardingReplicaReadWaitThroughput	• DBClusterIdentifier ,Role
ForwardingReplicaSelectLatency	• DBClusterIdentifier ,Role

Metric	Dimensions
ForwardingReplicaSelectThroughput	• DBClusterIdentifier ,Role
FreeLocalStorage	• DBClusterIdentifier ,Role
FreeStorageSpace	• DBClusterIdentifier
FreeableMemory	• DBClusterIdentifier • DBClusterIdentifier ,Role
InsertLatency	• DBClusterIdentifier ,Role
InsertThroughput	• DBClusterIdentifier ,Role
LoginFailures	• DBClusterIdentifier ,Role
NetworkReceiveThroughput	• DBClusterIdentifier ,Role
NetworkThroughput	• DBClusterIdentifier ,Role
NetworkTransmitThroughput	• DBClusterIdentifier ,Role
NumBinaryLogFiles	• DBClusterIdentifier ,Role
PurgeBoundary	• DBClusterIdentifier ,Role
PurgeFinishedPoint	• DBClusterIdentifier ,Role
Queries	• DBClusterIdentifier ,Role
ReadIOPS	• DBClusterIdentifier • DBClusterIdentifier ,Role
ReadLatency	• DBClusterIdentifier • DBClusterIdentifier ,Role
ReadThroughput	• DBClusterIdentifier • DBClusterIdentifier ,Role

Metric	Dimensions
ResultSetCacheHitRatio	• DBClusterIdentifier ,Role
RollbackSegmentHistoryListLength	• DBClusterIdentifier ,Role
RowLockTime	• DBClusterIdentifier ,Role
SelectLatency	• DBClusterIdentifier ,Role
SelectThroughput	• DBClusterIdentifier ,Role
StorageNetworkReceiveThroughput	• DBClusterIdentifier ,Role
StorageNetworkThroughput	• DBClusterIdentifier ,Role
StorageNetworkTransmitThroughput	• DBClusterIdentifier ,Role
SumBinaryLogSize	• DBClusterIdentifier ,Role
SwapUsage	• DBClusterIdentifier ,Role
TruncateFinishedPoint	• DBClusterIdentifier ,Role
UpdateLatency	• DBClusterIdentifier ,Role
UpdateThroughput	• DBClusterIdentifier ,Role
VolumeBytesUsed	• DbClusterIdentifier ,EngineName
VolumeReadIOPs	• DbClusterIdentifier ,EngineName
VolumeWriteIOPs	• DbClusterIdentifier ,EngineName
WriteIOPS	• DBClusterIdentifier • DBClusterIdentifier ,Role
WriteLatency	• DBClusterIdentifier • DBClusterIdentifier ,Role

Metric	Dimensions
WriteThroughput	• DBClusterIdentifier • DBClusterIdentifier ,Role

AWS::RDS::DBInstance

Metric	Dimensions
ACUUtilization	• DBInstanceIdentifier
AuroraEstimatedSharedMemoryBytes	• DBInstanceIdentifier
AuroraReplicaLag	• DBInstanceIdentifier
BurstBalance	• DBInstanceIdentifier
CPUUtilization	• DBInstanceIdentifier
DBLoad	• DBInstanceIdentifier
DBLoadCPU	• DBInstanceIdentifier
DBLoadNonCPU	• DBInstanceIdentifier
DBLoadRelativeToNumVCPUs	• DBInstanceIdentifier
DatabaseConnections	• DBInstanceIdentifier
FreeStorageSpace	• DBInstanceIdentifier
FreeableMemory	• DBInstanceIdentifier
MaximumUsedTransactionIDs	• DBInstanceIdentifier
OldestReplicationSlotLag	• DBInstanceIdentifier
ReadIOPS	• DBInstanceIdentifier
ReadLatency	• DBInstanceIdentifier

Metric	Dimensions
ReadThroughput	• DBInstanceIdentifier
ReplicaLag	• DBInstanceIdentifier
TransactionLogsDiskUsage	• DBInstanceIdentifier
TransactionLogsGeneration	• DBInstanceIdentifier
WriteIOPS	• DBInstanceIdentifier
WriteLatency	• DBInstanceIdentifier
WriteThroughput	• DBInstanceIdentifier

AWS/RUM

The AWS/RUM namespace includes enriched metrics for the following resource types.

- AWS::RUM::AppMonitor

AWS::RUM::AppMonitor

Metric	Dimensions
Http4xxCount	• application_name
Http5xxCount	• application_name
HttpStatusCodeCount	• application_name ,event_details.response.status ,event_type
JsErrorCount	• application_name
NavigationFrustratedTransaction	• application_name
NavigationSatisfiedTransaction	• application_name

Metric	Dimensions
NavigationToleratedTransaction	• application_name
PageViewCount	• application_name
PerformanceNavigationDuration	• application_name
PerformanceResourceDuration	• application_name ,event_details.file.type ,event_type
RumEventPayloadSize	• application_name
SessionCount	• application_name
WebVitalsCumulativeLayoutShift	• application_name
WebVitalsFirstInputDelay	• application_name
WebVitalsLargestContentfulPaint	• application_name

AWS/Redshift

The AWS/Redshift namespace includes enriched metrics for the following resource types.

- AWS::Redshift::Cluster

AWS::Redshift::Cluster

Metric	Dimensions
CPUUtilization	• ClusterIdentifier • ClusterIdentifier ,NodeID
CommitQueueLength	• ClusterIdentifier • ClusterIdentifier ,NodeID
ConcurrencyScalingActiveClusters	• ClusterIdentifier

Metric	Dimensions
ConcurrencyScalingSeconds	• ClusterIdentifier
DatabaseConnections	• ClusterIdentifier • ClusterIdentifier ,NodeID
HealthStatus	• ClusterIdentifier • ClusterIdentifier ,NodeID
MaintenanceMode	• ClusterIdentifier • ClusterIdentifier ,NodeID
MaxConfiguredConcurrencyScalingClusters	• ClusterIdentifier
NetworkReceiveThroughput	• ClusterIdentifier • ClusterIdentifier ,NodeID
NetworkTransmitThroughput	• ClusterIdentifier • ClusterIdentifier ,NodeID
NumExceededSchemaQuotas	• ClusterIdentifier
PercentageDiskSpaceUsed	• ClusterIdentifier • ClusterIdentifier ,NodeID
QueriesCompletedPerSecond	• ClusterIdentifier ,NodeID, latency • ClusterIdentifier ,latency
QueryDuration	• ClusterIdentifier ,NodeID, latency • ClusterIdentifier ,latency
QueryRuntimeBreakdown	• ClusterIdentifier ,NodeID, stage • ClusterIdentifier ,stage

Metric	Dimensions
ReadIOPS	• ClusterIdentifier • ClusterIdentifier ,NodeID
ReadLatency	• ClusterIdentifier • ClusterIdentifier ,NodeID
ReadThroughput	• ClusterIdentifier • ClusterIdentifier ,NodeID
RedshiftManagedStorageTotal Capacity	• ClusterIdentifier
TotalTableCount	• ClusterIdentifier • ClusterIdentifier ,NodeID
WLMQueriesCompletedPerSecond	• ClusterIdentifier ,NodeID, wlmid • ClusterIdentifier ,QueueName • ClusterIdentifier ,wlmid
WLMQueryDuration	• ClusterIdentifier ,NodeID, wlmid • ClusterIdentifier ,QueueName • ClusterIdentifier ,wlmid
WLMQueueLength	• ClusterIdentifier • ClusterIdentifier ,NodeID • ClusterIdentifier ,NodeID, QueueName • ClusterIdentifier ,NodeID, service class • ClusterIdentifier ,QueueName • ClusterIdentifier ,service class

Metric	Dimensions
WLMQueueWaitTime	• ClusterIdentifier ,QueryPriority • ClusterIdentifier ,QueueName • ClusterIdentifier ,wlmid
WLMRunningQueries	• ClusterIdentifier • ClusterIdentifier ,QueueName • ClusterIdentifier ,wlmid
WriteIOPS	• ClusterIdentifier • ClusterIdentifier ,NodeID
WriteLatency	• ClusterIdentifier • ClusterIdentifier ,NodeID
WriteThroughput	• ClusterIdentifier • ClusterIdentifier ,NodeID

AWS/Route53

The AWS/Route53 namespace includes enriched metrics for the following resource types.

- AWS::Route53::HealthCheck

AWS::Route53::HealthCheck

Metric	Dimensions
ChildHealthCheckHealthyCount	• HealthCheckId
ConnectionTime	• HealthCheckId • HealthCheckId ,Region
HealthCheckPercentageHealthy	• HealthCheckId

Metric	Dimensions
HealthCheckStatus	• HealthCheckId
SSLHandshakeTime	• HealthCheckId • HealthCheckId ,Region
TimeToFirstByte	• HealthCheckId • HealthCheckId ,Region

AWS/Route53Resolver

The AWS/Route53Resolver namespace includes enriched metrics for the following resource types.

- AWS::Route53Resolver::FirewallRuleGroup
- AWS::Route53Resolver::ResolverEndpoint

AWS::Route53Resolver::FirewallRuleGroup

Metric	Dimensions
FirewallRuleGroupQueryVolume	• FirewallRuleGroupId
FirewallRuleGroupVpcQueryVolume	• FirewallRuleGroupId ,VpcId

AWS::Route53Resolver::ResolverEndpoint

Metric	Dimensions
AverageResponseTime	• EndpointId ,RniId
CapacityUsage_BETA	• EndpointId ,RniId
ConntrackUtilization	• EndpointId ,RniId
EndpointHealthyENICount	• EndpointId

Metric	Dimensions
EndpointUnhealthyENICount	• EndpointId
FormErrCount	• EndpointId ,RniId
FormErrorQueries	• EndpointId ,RniId
InboundQueryVolume	• EndpointId
NXDomainCount	• EndpointId ,RniId
NoErrorCount	• EndpointId ,RniId
NxDomainQueries	• EndpointId ,RniId
OutboundQueryAggregateVolume	• EndpointId
OutboundQueryVolume	• EndpointId
P90ResponseTime	• EndpointID ,TargetNameServerIP • EndpointId ,RniId
RefusedCount	• EndpointId ,RniId
RefusedQueries	• EndpointId ,RniId
ReplyCount	• EndpointId ,RniId
RequestQueries	• EndpointID ,TargetNameServerIP
ResolverEndpointCapacityStatus	• EndpointId
ServFailQueries	• EndpointId ,RniId
ServfailCount	• EndpointId ,RniId
TimeoutCount	• EndpointId ,RniId
TimeoutQueries	• EndpointID ,TargetNameServerIP • EndpointId ,RniId

Metric	Dimensions
tcpRequestCount	• EndpointId , RniId
udpRequestCount	• EndpointId , RniId

AWS/S3

The AWS/S3 namespace includes enriched metrics for the following resource types.

- AWS::S3::Bucket

AWS::S3::Bucket

Metric	Dimensions
4xxErrors	• BucketName , FilterId
5xxErrors	• BucketName , FilterId
AllRequests	• BucketName , FilterId
BucketSizeBytes	• BucketName , StorageType
BytesDownloaded	• BucketName , FilterId
BytesPendingReplication	• DestinationBucket , RuleId, SourceBucket
BytesUploaded	• BucketName , FilterId
DeleteRequests	• BucketName , FilterId
FirstByteLatency	• BucketName , FilterId
GetRequests	• BucketName , FilterId
HeadRequests	• BucketName , FilterId

Metric	Dimensions
ListRequests	• BucketName , FilterId
NumberOfObjects	• BucketName , StorageType
OperationsFailedReplication	• DestinationBucket , RuleId, SourceBucket
OperationsPendingReplication	• DestinationBucket , RuleId, SourceBucket
PostRequests	• BucketName , FilterId
PutRequests	• BucketName , FilterId
ReplicationLatency	• DestinationBucket , RuleId, SourceBucket
TotalRequestLatency	• BucketName , FilterId

AWS/SNS

The AWS/SNS namespace includes enriched metrics for the following resource types.

- AWS::SNS::Topic

AWS::SNS::Topic

Metric	Dimensions
NumberOfMessagesPublished	• TopicName
NumberOfNotificationsDelivered	• TopicName
NumberOfNotificationsFailed	• TopicName
PublishSize	• TopicName

AWS/SQS

The AWS/SQS namespace includes enriched metrics for the following resource types.

- AWS::SQS::Queue

AWS::SQS::Queue

Metric	Dimensions
ApproximateAgeOfOldestMessage	• QueueName
ApproximateAgeOfOldestMessageInQuietGroups	• QueueName
ApproximateNumberOfGroupsWithInflightMessages	• QueueName
ApproximateNumberOfMessagesDelayed	• QueueName
ApproximateNumberOfMessagesDelayedInQuietGroups	• QueueName
ApproximateNumberOfMessagesNotVisible	• QueueName
ApproximateNumberOfMessagesNotVisibleInQuietGroups	• QueueName
ApproximateNumberOfMessagesVisible	• QueueName
ApproximateNumberOfMessagesVisibleInQuietGroups	• QueueName
ApproximateNumberOfNoisyGroups	• QueueName
NumberOfDeduplicatedSentMessages	• QueueName

Metric	Dimensions
NumberOfEmptyReceives	• QueueName
NumberOfMessagesDeleted	• QueueName
NumberOfMessagesReceived	• QueueName
NumberOfMessagesSent	• QueueName
SentMessageSize	• QueueName

AWS/SageMaker

The AWS/SageMaker namespace includes enriched metrics for the following resource types.

- AWS::SageMaker::Endpoint
- AWS::SageMaker::InferenceComponent

AWS::SageMaker::Endpoint

Metric	Dimensions
ConcurrentRequestsPerCopy	• EndpointName , VariantName
ConcurrentRequestsPerModel	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • EndpointName , VariantName
FirstChunkLatency	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • EndpointName , VariantName
FirstChunkModelLatency	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • EndpointName , VariantName

Metric	Dimensions
FirstChunkOverheadLatency	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • EndpointName , VariantName
Invocation4XXErrors	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName
Invocation5XXErrors	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName
InvocationModelErrors	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName

Metric	Dimensions
Invocations	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName
InvocationsPerCopy	• EndpointName , VariantName
InvocationsPerInstance	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName
MidStreamErrors	• EndpointName , VariantName
ModelCacheHit	• EndpointName , VariantName
ModelDownloadingTime	• EndpointName , VariantName
ModelLatency	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName
ModelLoadingTime	• EndpointName , VariantName
ModelLoadingWaitTime	• EndpointName , VariantName

Metric	Dimensions
ModelSetupTime	• EndpointName , VariantName
ModelUnloadingTime	• EndpointName , VariantName
OverheadLatency	• AvailabilityZone , EndpointName , InstanceType , Region, VariantName • ContainerId , EndpointName , InferenceComponentName , InstanceId , VariantName • EndpointName , InstanceId , VariantName • EndpointName , VariantName

AWS::SageMaker::InferenceComponent

Metric	Dimensions
ConcurrentRequestsPerCopy	• AvailabilityZone , Inference ComponentName , InstanceType , Region • InferenceComponentName
ConcurrentRequestsPerModel	• InferenceComponentName
Invocation4XXErrors	• AvailabilityZone , Inference ComponentName , InstanceType , Region • InferenceComponentName
Invocation5XXErrors	• AvailabilityZone , Inference ComponentName , InstanceType , Region • InferenceComponentName

Metric	Dimensions
InvocationModelErrors	• AvailabilityZone ,InferenceComponentName ,InstanceType ,Region • InferenceComponentName
Invocations	• AvailabilityZone ,InferenceComponentName ,InstanceType ,Region • InferenceComponentName
InvocationsPerCopy	• AvailabilityZone ,InferenceComponentName ,InstanceType ,Region • InferenceComponentName
InvocationsPerInstance	• InferenceComponentName
ModelLatency	• AvailabilityZone ,InferenceComponentName ,InstanceType ,Region • InferenceComponentName
ModelSetupTime	• InferenceComponentName
OverheadLatency	• AvailabilityZone ,InferenceComponentName ,InstanceType ,Region • InferenceComponentName

AWS/Scheduler

The AWS/Scheduler namespace includes enriched metrics for the following resource types.

- AWS::Scheduler::ScheduleGroup

AWS::Scheduler::ScheduleGroup

Metric	Dimensions
InvocationAttemptCount	• ScheduleGroup
InvocationDroppedCount	• ScheduleGroup
InvocationThrottleCount	• ScheduleGroup
InvocationsFailedToBeSentToDeadLetterCount	• ScheduleGroup
InvocationsFailedToBeSentToDeadLetterCount_AWS.SimpleQueueService.NonExistentQueue	• ScheduleGroup
InvocationsFailedToBeSentToDeadLetterCount_AccessDenied	• ScheduleGroup
InvocationsFailedToBeSentToDeadLetterCount_KMS.AccessDeniedException	• ScheduleGroup
InvocationsSentToDeadLetterCount	• ScheduleGroup
TargetErrorCount	• ScheduleGroup
TargetErrorThrottledCount	• ScheduleGroup

AWS/Transfer

The AWS/Transfer namespace includes enriched metrics for the following resource types.

- AWS::Transfer::Connector
- AWS::Transfer::Server

AWS::Transfer::Connector

Metric	Dimensions
OutboundFailedMessage	• ConnectorId
OutboundMessage	• ConnectorId

AWS::Transfer::Server

Metric	Dimensions
BytesIn	• ServerId
BytesOut	• ServerId
ConcurrentSessions	• ServerId
FilesIn	• ServerId
FilesOut	• ServerId
InboundFailedMessage	• ServerId
InboundMessage	• ServerId
OnUploadExecutionsFailed	• ServerId
OnUploadExecutionsStarted	• ServerId
OnUploadExecutionsSuccess	• ServerId
OutboundFailedMessage	• ServerId

AWS/TransitGateway

The AWS/TransitGateway namespace includes enriched metrics for the following resource types.

- AWS::EC2::TransitGateway

AWS::EC2::TransitGateway

Metric	Dimensions
BytesDropCountBlackhole	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
BytesDropCountInternalError	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
BytesDropCountNoPolicy	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
BytesDropCountNoRoute	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
BytesIn	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment

Metric	Dimensions
	• TransitGateway • TransitGateway ,TransitGa tewayAttachment
BytesOut	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketDropCountBlackhole	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketDropCountInternalError	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketDropCountNoPolicy	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment

Metric	Dimensions
PacketDropCountNoRoute	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketDropCountTTLExpired	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketsIn	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment
PacketsOut	• AvailabilityZone ,TransitGateway • AvailabilityZone ,TransitGateway , TransitGatewayAttachment • TransitGateway • TransitGateway ,TransitGa tewayAttachment

AWS/VPN

The AWS/VPN namespace includes enriched metrics for the following resource types.

- AWS::EC2::VPNConnection

AWS::EC2::VPNConnection

Metric	Dimensions
TunnelDataIn	• VpnId
TunnelDataOut	• VpnId
TunnelState	• VpnId

AWS/VpcLattice

The AWS/VpcLattice namespace includes enriched metrics for the following resource types.

- AWS::VpcLattice::Service

AWS::VpcLattice::Service

Metric	Dimensions
HTTPCode_2XX_Count	• AvailabilityZone ,Service
HTTPCode_3XX_Count	• AvailabilityZone ,Service
HTTPCode_4XX_Count	• AvailabilityZone ,Service
HTTPCode_5XX_Count	• AvailabilityZone ,Service
RequestTime	• AvailabilityZone ,Service
RequestTimeoutCount	• AvailabilityZone ,Service
TotalRequestCount	• AvailabilityZone ,Service

AWS/WorkSpaces

The AWS/WorkSpaces namespace includes enriched metrics for the following resource types.

- AWS::WorkSpaces::Workspace

AWS::WorkSpaces::Workspace

Metric	Dimensions
Available	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
CPUUsage	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
ConnectionAttempt	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
ConnectionFailure	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId

Metric	Dimensions
	• WorkspaceId
ConnectionSuccess	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
InSessionLatency	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
Maintenance	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
MemoryUsage	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId

Metric	Dimensions
Restoring	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
RootVolumeDiskUsage	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
SessionDisconnect	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
SessionLaunchTime	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId

Metric	Dimensions
Stopped	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
UDPPacketLossRate	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
Unhealthy	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
UpTime	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId

Metric	Dimensions
UserConnected	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId
UserVolumeDiskUsage	• AwsAccountId , BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • AwsAccountId , WorkspaceId • BundleId, ComputeType , Protocol, RunningMode , WorkspaceId • WorkspaceId

CloudWatchSynthetics

The CloudWatchSynthetics namespace includes enriched metrics for the following resource types.

- AWS::Synthetics::Canary

AWS::Synthetics::Canary

Metric	Dimensions
2xx	• CanaryName
4xx	• CanaryName
5xx	• CanaryName
Duration	• CanaryName

Metric	Dimensions
Error	• CanaryName
Failed	• CanaryName
Failed requests	• CanaryName
SuccessPercent	• CanaryName
VisualMonitoringSuccessPercent	• CanaryName
VisualMonitoringTotalComparisons	• CanaryName

ContainerInsights

The ContainerInsights namespace includes enriched metrics for the following resource types.

- AWS::EKS::Cluster

AWS::EKS::Cluster

Metric	Dimensions
cluster_failed_node_count	• ClusterName
cluster_node_count	• ClusterName
node_cpu_limit	• ClusterName
node_cpu_usage_total	• ClusterName
node_filesystem_utilization	• ClusterName
node_memory_limit	• ClusterName
node_memory_working_set	• ClusterName
node_network_total_bytes	• ClusterName

Metric	Dimensions
pod_cpu_utilization	• ClusterName
pod_network_rx_bytes	• ClusterName
pod_network_tx_bytes	• ClusterName

ECS/ContainerInsights

The ECS/ContainerInsights namespace includes enriched metrics for the following resource types.

- AWS::ECS::Cluster
- AWS::ECS::Service

AWS::ECS::Cluster

Metric	Dimensions
ContainerInstanceCount	• ClusterName
CpuReserved	• ClusterName • ClusterName ,TaskDefinitionFamily
CpuUtilized	• ClusterName • ClusterName ,TaskDefinitionFamily
EphemeralStorageReserved	• ClusterName • ClusterName ,TaskDefinitionFamily
EphemeralStorageUtilized	• ClusterName • ClusterName ,TaskDefinitionFamily

Metric	Dimensions
MemoryReserved	• ClusterName • ClusterName ,TaskDefinitionFamily
MemoryUtilized	• ClusterName • ClusterName ,TaskDefinitionFamily
NetworkRxBytes	• ClusterName • ClusterName ,TaskDefinitionFamily
NetworkTxBytes	• ClusterName • ClusterName ,TaskDefinitionFamily
ServiceCount	• ClusterName
StorageReadBytes	• ClusterName ,TaskDefinitionFamily
StorageWriteBytes	• ClusterName ,TaskDefinitionFamily
TaskCount	• ClusterName

AWS::ECS::Service

Metric	Dimensions
CpuReserved	• ClusterName ,ServiceName
CpuUtilized	• ClusterName ,ServiceName
DeploymentCount	• ClusterName ,ServiceName

Metric	Dimensions
DesiredTaskCount	• ClusterName , ServiceName
EphemeralStorageReserved	• ClusterName , ServiceName
EphemeralStorageUtilized	• ClusterName , ServiceName
MemoryReserved	• ClusterName , ServiceName
MemoryUtilized	• ClusterName , ServiceName
NetworkRxBytes	• ClusterName , ServiceName
NetworkTxBytes	• ClusterName , ServiceName
PendingTaskCount	• ClusterName , ServiceName
RunningTaskCount	• ClusterName , ServiceName
TaskSetCount	• ClusterName , ServiceName

Glue

The Glue namespace includes enriched metrics for the following resource types.

- AWS::Glue::Job

AWS::Glue::Job

Metric	Dimensions
glue.ALL.jvm.heap.usage	• JobName, JobRunId, Type
glue.ALL.jvm.heap.used	• JobName, JobRunId, Type
glue.ALL.s3.filesystem.read_bytes	• JobName, JobRunId, Type

Metric	Dimensions
glue.ALL.s3.filesystem.write_bytes	• JobName, JobRunId, Type
glue.ALL.system.cpuSystemLoad	• JobName, JobRunId, Type
glue.driver.BlockManager.disk.diskSpaceUsed_MB	• JobName, JobRunId, Type
glue.driver.ExecutorAllocationManager.executors.numberAllExecutors	• JobName, JobRunId, Type
glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors	• JobName, JobRunId, Type
glue.driver.aggregate.bytesRead	• JobName, JobRunId, Type
glue.driver.aggregate.elapsedTime	• JobName, JobRunId, Type
glue.driver.aggregate.numCompletedStages	• JobName, JobRunId, Type
glue.driver.aggregate.numCompletedTasks	• JobName, JobRunId, Type
glue.driver.aggregate.numFailedTasks	• JobName, JobRunId, Type
glue.driver.aggregate.numKilledTasks	• JobName, JobRunId, Type
glue.driver.aggregate.recordsRead	• JobName, JobRunId, Type

Metric	Dimensions
glue.driver.aggregate.shuffleBytesWritten	• JobName, JobRunId, Type
glue.driver.aggregate.shuffleLocalBytesRead	• JobName, JobRunId, Type
glue.driver.jvm.heap.usage	• JobName, JobRunId, Type
glue.driver.jvm.heap.used	• JobName, JobRunId, Type
glue.driver.s3.filesystem.read_bytes	• JobName, JobRunId, Type
glue.driver.s3.filesystem.write_bytes	• JobName, JobRunId, Type
glue.driver.streaming.batchProcessingTimeInMs	• JobName, JobRunId, Type
glue.driver.streaming.numRecords	• JobName, JobRunId, Type
glue.driver.system.cpuSystemLoad	• JobName, JobRunId, Type

LambdaInsights

The LambdaInsights namespace includes enriched metrics for the following resource types.

- AWS::Lambda::Function

AWS::Lambda::Function

Metric	Dimensions
cpu_total_time	• function_name
init_duration	• function_name

Metric	Dimensions
memory_utilization	• function_name
rx_bytes	• function_name
total_memory	• function_name
total_network	• function_name
tx_bytes	• function_name
used_memory_max	• function_name

Disable resource tags on telemetry

If you don't need resource tags for telemetry, disable the feature. When disabled, CloudWatch stops enriching telemetry with tags. You can enable it again at any time. For more information, see Enable resource tags on telemetry.

Verify you have permissions to disable resource tags for telemetry.

> **Note**
>
> To disable resource tags on telemetry, you must be signed in to an IAM principal that has the `observabilityadmin:StopTelemetryEnrichment` and `resource-explorer-2:DeleteStreamingAccessForService` permissions.

To disable resource tags for telemetry (CloudWatch Console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **CloudWatch**, then choose **Settings**.
3. In the **Enable resource tags for telemetry** pane, choose **off**.
4. In the confirm modal, read through the consequences of disabling resource tags for telemetry, then type **`confirm`** and choose **Disable resource tags**.

> **Note**
>
> In the CloudWatch console, you must be signed in to an IAM principal that has the `observabilityadmin:GetTelemetryEnrichmentStatus` permission.

To disable resource tags for telemetry (AWS CLI)

Use the `stop-telemetry-enrichment` command to disable resource tags for telemetry.

```
aws observabilityadmin stop-telemetry-enrichment
```

After you complete these steps, CloudWatch stops enriching telemetry with tags. Metrics previously enriched with resource tags can still be discovered for up to 14 days. Logs previously enriched with resource tags can still be queried until the log group's retention period expires.

To disable resource tags for telemetry (AWS CloudFormation)

If you enabled resource tags for telemetry by using an AWS::ObservabilityAdmin::TelemetryEnrichment resource, disable the feature by removing that resource from your CloudFormation template and updating the stack.

Troubleshooting

CloudWatch cannot enable or disable resource tags for telemetry

If you receive an error when trying to enable or disable resource tags for telemetry, verify that the required roles and permissions are correctly configured.

To enable resource tags on telemetry, you must be signed in to an IAM principal that has the `observabilityadmin:StartTelemetryEnrichment`, `iam:CreateServiceLinkedRole`, `resource-explorer-2:CreateIndex`, `resource-explorer-2:CreateManagedView` and `resource-explorer-2:CreateStreamingAccessForService` permissions.

To disable resource tags on telemetry, you must be signed in to an IAM principal that has the `observabilityadmin:StopTelemetryEnrichment` and `resource-explorer-2:DeleteStreamingAccessForService` permissions.

> **Note**
>
> In the CloudWatch console, you must be signed in to an IAM principal that has the `observabilityadmin:GetTelemetryEnrichmentStatus` permission.

Telemetry enrichment status is `Impaired`

When the status shows as Impaired, the enable/disable request failed to complete. Retry your request to achieve the desired Running or Stopped status.

Missing tags after enabling telemetry enrichment

After you enable the feature, CloudWatch begins enriching telemetry with tags. CloudWatch can take up to 3 hours to discover all your tags for enrichment.

I turned off resource tags for telemetry, I can still see tags enriching my telemetry

After you disable the feature:

- Metrics previously enriched with resource tags can still be discovered for up to 14 days.
- Logs previously enriched with resource tags can still be queried until the log group's retention period expires.

CloudWatch Metrics (Classic)

CloudWatch Metrics (Classic) uses the `PutMetricData` API and embedded metric format (EMF) for ingestion, and provides CloudWatch Metrics Insights (SQL-based queries) for analysis. Classic metrics support up to 30 dimensions per metric with per-metric-per-month pricing.

Use Classic metrics when you have existing integrations with the CloudWatch API, need compatibility with AWS service metrics that are not yet available in OpenTelemetry format, or prefer SQL-based querying with CloudWatch Metrics Insights.

Topics

- Metrics concepts
- Basic monitoring and detailed monitoring in CloudWatch
- Publish custom metrics (PutMetricData / EMF)

- Query your CloudWatch metrics with CloudWatch Metrics Insights
- View available metrics
- Retrieve metric data (GetMetricData)
- Get statistics for a metric (GetMetricStatistics)
- Use metrics explorer to monitor resources by their tags and properties
- Use search expressions in graphs
- Use metric streams
- Math expressions with metrics
- Using CloudWatch anomaly detection
- Query metrics from other data sources
- Grafana integration
- AWS services that publish CloudWatch metrics

Metrics concepts

The following terminology and concepts are central to your understanding and use of Amazon CloudWatch:

- Namespaces
- Metrics
- Dimensions
- Resolution
- Statistics
- Percentiles
- Alarms

For information about the service quotas for CloudWatch metrics, alarms, API requests, and alarm email notifications, see CloudWatch service quotas.

Namespaces

A *namespace* is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

There is no default namespace. You must specify a namespace for each data point you publish to CloudWatch. You can specify a namespace name when you create a metric. These names must contain valid ASCII characters, and be 255 or fewer characters. Possible characters are: alphanumeric characters (0-9A-Za-z), period (.), hyphen (-), underscore (_), forward slash (/), hash (#), colon (:), and the space character. A namespace must contain at least one non-whitespace character.

The AWS namespaces typically use the following naming convention: `AWS/service`. For example, Amazon EC2 uses the `AWS/EC2` namespace. For the list of AWS namespaces, see AWS services that publish CloudWatch metrics.

Metrics

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Think of a metric as a variable to monitor, and the data points as representing the values of that variable over time. For example, the CPU usage of a particular EC2 instance is one metric provided by Amazon EC2. The data points themselves can come from any application or business activity from which you collect data.

By default, many AWS services provide metrics at no charge for resources (such as Amazon EC2 instances, Amazon EBS volumes, and Amazon RDS DB instances). For a charge, you can also enable detailed monitoring for some resources, such as your Amazon EC2 instances, or publish your own application metrics. For custom metrics, you can add the data points in any order, and at any rate you choose. You can retrieve statistics about those data points as an ordered set of time-series data.

Metrics exist only in the Region in which they are created. Metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them. Data points older than 15 months expire on a rolling basis; as new data points come in, data older than 15 months is dropped.

Metrics are uniquely defined by a name, a namespace, and zero or more dimensions. Each data point in a metric has a time stamp, and (optionally) a unit of measure. You can retrieve statistics from CloudWatch for any metric.

For more information, see View available metrics and Publish custom metrics (PutMetricData / EMF).

Time stamps

Each metric data point must be associated with a time stamp. The time stamp can be up to two weeks in the past and up to two hours into the future. If you do not provide a time stamp, CloudWatch creates a time stamp for you based on the time the data point was received.

Time stamps are `dateTime` objects, with the complete date plus hours, minutes, and seconds (for example, 2016-10-31T23:59:59Z). For more information, see dateTime. Although it is not required, we recommend that you use UTC. When you retrieve statistics from CloudWatch, all times are in UTC.

CloudWatch alarms check metrics based on the current time in UTC. Custom metrics sent to CloudWatch with time stamps other than the current UTC time can cause alarms to display the **Insufficient Data** state or result in delayed alarms.

Metrics retention

CloudWatch retains metric data as follows:

- Data points with a period of less than 60 seconds are available for 3 hours. These data points are high-resolution custom metrics.
- Data points with a period of 60 seconds (1 minute) are available for 15 days
- Data points with a period of 300 seconds (5 minutes) are available for 63 days
- Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)

Data points that are initially published with a shorter period are aggregated together for long-term storage. For example, if you collect data using a period of 1 minute, the data remains available for 15 days with 1-minute resolution. After 15 days this data is still available, but is aggregated and is retrievable only with a resolution of 5 minutes. After 63 days, the data is further aggregated and is available with a resolution of 1 hour.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **Browse** tab in the console. They are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the AWS CLI.

Dimensions

A *dimension* is a name/value pair that is part of the identity of a metric. You can assign up to 30 dimensions to a metric.

Every metric has specific characteristics that describe it, and you can think of dimensions as categories for those characteristics. Dimensions help you design a structure for your statistics plan. Because dimensions are part of the unique identifier for a metric, whenever you add a unique name/value pair to one of your metrics, you are creating a new variation of that metric.

AWS services that send data to CloudWatch attach dimensions to each metric. You can use dimensions to filter the results that CloudWatch returns. For example, you can get statistics for a specific EC2 instance by specifying the `InstanceId` dimension when you search for metrics.

For metrics produced by certain AWS services, such as Amazon EC2, CloudWatch can aggregate data across dimensions. For example, if you search for metrics in the `AWS/EC2` namespace but do not specify any dimensions, CloudWatch aggregates all data for the specified metric to create the statistic that you requested. CloudWatch does not aggregate across dimensions for your custom metrics.

Dimension combinations

CloudWatch treats each unique combination of dimensions as a separate metric, even if the metrics have the same metric name. You can only retrieve statistics using combinations of dimensions that you specifically published. When you retrieve statistics, specify the same values for the namespace, metric name, and dimension parameters that were used when the metrics were created. You can also specify the start and end times for CloudWatch to use for aggregation.

For example, suppose that you publish four distinct metrics named ServerStats in the DataCenterMetric namespace with the following properties:

```
Dimensions: Server=Prod, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:30:00Z, Value: 105
Dimensions: Server=Beta, Domain=Frankfurt, Unit: Count, Timestamp:
 2016-10-31T12:31:00Z, Value: 115
Dimensions: Server=Prod, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:32:00Z, Value: 95
Dimensions: Server=Beta, Domain=Rio,       Unit: Count, Timestamp:
 2016-10-31T12:33:00Z, Value: 97
```

If you publish only those four metrics, you can retrieve statistics for these combinations of dimensions:

- `Server=Prod,Domain=Frankfurt`
- `Server=Prod,Domain=Rio`
- `Server=Beta,Domain=Frankfurt`
- `Server=Beta,Domain=Rio`

You can't retrieve statistics for the following dimensions or if you specify no dimensions. (The exception is by using the metric math **SEARCH** function, which can retrieve statistics for multiple metrics. For more information, see Use search expressions in graphs.)

- `Server=Prod`
- `Server=Beta`
- `Domain=Frankfurt`
- `Domain=Rio`

> **Note**
>
> OpenTelemetry metrics use labels instead of dimensions. Labels serve a similar purpose but follow OpenTelemetry semantic conventions and support up to 150 labels per metric. For more information, see OpenTelemetry Metrics (Recommended).

Resolution

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

> **Note**
>
> OpenTelemetry metrics are ingested at the resolution they are sent, with no minimum granularity restriction. Resolution and rollup behavior for OpenTelemetry metrics differs from traditional CloudWatch metrics.

Statistics

Statistics are metric data aggregations over specified periods of time. CloudWatch provides statistics based on the metric data points provided by your custom data or provided by other AWS services to CloudWatch. Aggregations are made using the namespace, metric name, dimensions, and the data point unit of measure, within the time period you specify.

For detailed definitions of the statistics supported by CloudWatch, see CloudWatch statistics definitions.

Units

Each statistic has a unit of measure. Example units include `Bytes`, `Seconds`, `Count`, and `Percent`. For the complete list of the units that CloudWatch supports, see the MetricDatum data type in the *Amazon CloudWatch API Reference*.

You can specify a unit when you create a custom metric. If you do not specify a unit, CloudWatch uses None as the unit. Units help provide conceptual meaning to your data. Though CloudWatch attaches no significance to a unit internally, other applications can derive semantic information based on the unit.

Metric data points that specify a unit of measure are aggregated separately. When you get statistics without specifying a unit, CloudWatch aggregates all data points of the same unit

together. If you have two otherwise identical metrics with different units, two separate data streams are returned, one for each unit.

Periods

A *period* is the length of time associated with a specific Amazon CloudWatch statistic. Each statistic represents an aggregation of the metrics data collected for a specified period of time. Periods are defined in numbers of seconds, and valid values for period are 1, 5, 10, 30, or any multiple of 60. For example, to specify a period of six minutes, use 360 as the period value. You can adjust how the data is aggregated by varying the length of the period. The default value of a period is 60 seconds. A period can be as short as one second, and must be a multiple of 60 if it is greater than the default value of 60 seconds.

Only custom metrics that you define with a storage resolution of 1 second support sub-minute periods. Even though the option to set a period below 60 is always available in the console, you should select a period that aligns to how the metric is stored. For more information about metrics that support sub-minute periods, see High-resolution metrics.

When you retrieve statistics, you can specify a period, start time, and end time. These parameters determine the overall length of time associated with the statistics. The default values for the start time and end time get you the last hour's worth of statistics. The values that you specify for the start time and end time determine how many periods CloudWatch returns. For example, retrieving statistics using the default values for the period, start time, and end time returns an aggregated set of statistics for each minute of the previous hour. If you prefer statistics aggregated in ten-minute blocks, specify a period of 600. For statistics aggregated over the entire hour, specify a period of 3600.

When statistics are aggregated over a period of time, they are stamped with the time corresponding to the beginning of the period. For example, data aggregated from 7:00pm to 8:00pm is stamped as 7:00pm. Additionally, data aggregated between 7:00pm and 8:00pm begins to be visible at 7:00pm, then the values of that aggregated data may change as CloudWatch collects more samples during the period.

Periods are also important for CloudWatch alarms. When you create an alarm to monitor a specific metric, you are asking CloudWatch to compare that metric to the threshold value that you specified. You have extensive control over how CloudWatch makes that comparison. Not only can you specify the period over which the comparison is made, but you can also specify how many evaluation periods are used to arrive at a conclusion. For example, if you specify three evaluation

periods, CloudWatch compares a window of three data points. CloudWatch only notifies you if the oldest data point is breaching and the others are breaching or missing.

Aggregation

Amazon CloudWatch aggregates statistics according to the period length that you specify when retrieving statistics. You can publish as many data points as you want with the same or similar time stamps. CloudWatch aggregates them according to the specified period length. CloudWatch does not automatically aggregate data across Regions, but you can use metric math to aggregate metrics from different Regions.

You can publish data points for a metric that share not only the same time stamp, but also the same namespace and dimensions. CloudWatch returns aggregated statistics for those data points. You can also publish multiple data points for the same or different metrics, with any time stamp.

For large datasets, you can insert a pre-aggregated dataset called a *statistic set*. With statistic sets, you give CloudWatch the Min, Max, Sum, and SampleCount for a number of data points. This is commonly used when you need to collect data many times in a minute. For example, suppose you have a metric for the request latency of a web page. It doesn't make sense to publish data with every web page hit. Consider collecting the latency of all hits to that web page, aggregate them once a minute, and send that statistic set to CloudWatch.

Amazon CloudWatch doesn't differentiate the source of a metric. If you publish a metric with the same namespace and dimensions from different sources, CloudWatch treats this as a single metric. This can be useful for service metrics in a distributed, scaled system. For example, all the hosts in a web server application could publish identical metrics representing the latency of requests they are processing. CloudWatch treats these as a single metric, allowing you to get the statistics for minimum, maximum, average, and sum of all requests across your application.

Percentiles

A *percentile* indicates the relative standing of a value in a dataset. For example, the 95th percentile means that 95 percent of the data is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.

Percentiles are often used to isolate anomalies. In a normal distribution, 95 percent of the data is within two standard deviations from the mean and 99.7 percent of the data is within three standard deviations from the mean. Any data that falls outside three standard deviations is often considered to be an anomaly because it differs so greatly from the average value. For example,

suppose that you are monitoring the CPU utilization of your EC2 instances to make sure that your customers have a good experience. If you monitor the average, this can hide anomalies. If you monitor the maximum, a single anomaly can skew the results. Using percentiles, you can monitor the 95th percentile of CPU utilization to check for instances with an unusually heavy load.

Some CloudWatch metrics support percentiles as a statistic. For these metrics, you can monitor your system and applications using percentiles as you would when using the other CloudWatch statistics (Average, Minimum, Maximum, and Sum). For example, when you create an alarm, you can use percentiles as the statistical function. You can specify the percentile with up to ten decimal places (for example, p95.0123456789).

Percentile statistics are available for custom metrics as long as you publish the raw, unsummarized data points for your custom metric. Percentile statistics are not available for metrics when any of the metric values are negative numbers.

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can only retrieve percentile statistics for this data if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support percentile statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Lambda
- Amazon RDS

CloudWatch also supports trimmed mean and other performance statistics, which can have a similar use as percentiles. For more information, see CloudWatch statistics definitions.

Alarms

You can use an *alarm* to automatically initiate actions on your behalf. An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy. You can also add alarms to dashboards.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

When creating an alarm, select an alarm monitoring period that is greater than or equal to the metrics resolution. For example, basic monitoring for Amazon EC2 provides metrics for your instances every 5 minutes. When setting an alarm on a basic monitoring metric, select a period of at least 300 seconds (5 minutes). Detailed monitoring for Amazon EC2 provides metrics for your instances with a resolution of 1 minute. When setting an alarm on a detailed monitoring metric, select a period of at least 60 seconds (1 minute).

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see Publish custom metrics (PutMetricData / EMF).

For more information, see Using Amazon CloudWatch alarms and Create an alarm from a metric on a graph.

For OpenTelemetry metrics, you can create PromQL-based CloudWatch Alarms. These alarms use PromQL queries to define alarm conditions, using the same query language available in CloudWatch Query Studio.

Basic monitoring and detailed monitoring in CloudWatch

CloudWatch provides two categories of monitoring: *basic monitoring* and *detailed monitoring*.

Many AWS services offer basic monitoring by publishing a default set of metrics to CloudWatch with no charge to customers. By default, when you start using one of these AWS services, basic monitoring is automatically enabled. For a list of services that offer basic monitoring, see AWS services that publish CloudWatch metrics.

Detailed monitoring is offered by only some services. It also incurs charges. To use it for an AWS service, you must choose to activate it. For more information about pricing, see Amazon CloudWatch pricing.

Detailed monitoring options differ based on the services that offer it. For example, Amazon EC2 detailed monitoring provides more frequent metrics, published at one-minute intervals, instead of the five-minute intervals used in Amazon EC2 basic monitoring. Detailed monitoring for Amazon S3 and Amazon Managed Streaming for Apache Kafka means more fine-grained metrics.

In different AWS services, detailed monitoring also has different names. For example, in Amazon EC2 it is called detailed monitoring, in AWS Elastic Beanstalk it is called enhanced monitoring, and in Amazon S3 it is called request metrics.

Using detailed monitoring for Amazon EC2 helps you better manage your Amazon EC2 resources, so that you can find trends and take action faster. For Amazon S3 request metrics are available at one-minute intervals to help you quickly identify and act on operational issues. On Amazon MSK, when you enable the `PER_BROKER`, `PER_TOPIC_PER_BROKER`, or `PER_TOPIC_PER_PARTITION` level monitoring, you get additional metrics that provide more visibility.

The following table lists the services that offer detailed monitoring. It also includes links to the documentation for those services that explain more about the detailed monitoring and provide instructions for how to activate it.

Service	Documentation
Amazon API Gateway	Dimensions for API Gateway metrics
AWS AppSync	CloudWatch metrics
Amazon CloudFront	Viewing additional CloudFront distribution metrics
Amazon EC2	Manage detailed monitoring for your EC2 instances
Elastic Beanstalk	Enhanced health reporting and monitoring
Amazon Kinesis Data Streams	Enhanced Shard-level Metrics
AWS Lambda	Event source mapping metrics

Service	Documentation
Amazon MSK	Amazon MSK Metrics for Monitoring with CloudWatch
Amazon S3	Amazon S3 request metrics in CloudWatch
Amazon SES	Collect CloudWatch detailed monitoring metrics using Amazon SES event publishing.

Additionally, CloudWatch offers out-of-the-box monitoring solutions with more detailed metrics and pre-created dashboards for some AWS services, as shown in the following table.

Service	Feature documentation
Lambda	Lambda Insights
Amazon ECS	Container Insights for Amazon ECS
Amazon EKS	Container Insights for Amazon EKS and Kubernetes
Amazon RDS	CloudWatch Database Insights
Amazon Aurora	CloudWatch Database Insights

Publish custom metrics (PutMetricData / EMF)

You can publish your own custom metrics to CloudWatch using the `PutMetricData` API or the AWS CLI. This approach uses CloudWatch namespaces, metric names, and dimensions.

> **Note**
>
> For new implementations, we recommend using OpenTelemetry to publish custom metrics. For more information, see Publish custom metrics with OpenTelemetry.

Topics

- High-resolution metrics
- Use dimensions
- Publish single data points
- Publish statistic sets
- Publish the value zero
- Stop publishing metrics

High-resolution metrics

Each metric is one of the following:

- Standard resolution, with data having a one-minute granularity
- High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every `PutMetricData` call for a custom metric is charged, so calling `PutMetricData` more often on a high-resolution metric can lead to higher charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds or 30 seconds, or you can set a regular alarm with a period of any multiple of 60 seconds. There is a higher charge for high-resolution alarms with a period of 10 or 30 seconds.

Use dimensions

In custom metrics, the `--dimensions` parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 30 dimensions assigned to one metric, and each dimension is defined by a name and value pair.

How you specify a dimension is different when you use different commands. With put-metric-data, you specify each dimension as *MyName*=*MyValue*, and with get-metric-statistics or put-metric-

alarm you use the format Name=*MyName*, Value=*MyValue*. For example, the following command publishes a Buffers metric with 2 dimensions named InstanceId and InstanceType.

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit
 Bytes --value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

This command retrieves statistics for that same metric. Separate the Name and Value parts of a single dimension with commas, but if you have multiple dimensions, use a space between one dimension and the next.

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-
time 2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period
 60
```

If a single metric includes multiple dimensions, you must specify a value for every defined dimension when you use get-metric-statistics. For example, the Amazon S3 metric BucketSizeBytes includes the dimensions BucketName and StorageType, so you must specify both dimensions with get-metric-statistics.

```
aws cloudwatch get-metric-statistics --metric-name BucketSizeBytes --start-time
 2017-01-23T14:23:00Z --end-time 2017-01-26T19:30:00Z --period 3600 --namespace
 AWS/S3 --statistics Maximum --dimensions Name=BucketName,Value=amzn-s3-demo-bucket
 Name=StorageType,Value=StandardStorage --output table
```

To see what dimensions are defined for a metric, use the list-metrics command.

Publish single data points

To publish a single data point for a new or existing metric, use the put-metric-data command with one value and time stamp. For example, the following actions each publish one data point.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 2 --timestamp 2016-10-20T12:00:00.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 4 --timestamp 2016-10-20T12:00:01.000Z
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --
value 5 --timestamp 2016-10-20T12:00:02.000Z
```

If you call this command with a new metric name, CloudWatch creates a metric for you. Otherwise, CloudWatch associates your data with the existing metric that you specified.

> **Note**
>
> When you create a metric, it can take up to 2 minutes before you can retrieve statistics for the new metric using the get-metric-statistics command. However, it can take up to 15 minutes before the new metric appears in the list of metrics retrieved using the list-metrics command.

Although you can publish data points with time stamps as granular as one-thousandth of a second, CloudWatch aggregates the data to a minimum granularity of 1 second. CloudWatch records the average (sum of all items divided by number of items) of the values received for each period, as well as the number of samples, maximum value, and minimum value for the same time period. For example, the `PageViewCount` metric from the previous examples contains 3 data points with time stamps just seconds apart. If you have your period set to 1 minute, CloudWatch aggregates the 3 data points because they all have time stamps within a 1-minute period.

You can use the **get-metric-statistics** command to retrieve statistics based on the data points that you published.

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount
 \
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" \
--start-time 2016-10-20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

The following is example output.

```
{
    "Datapoints": [
        {
            "SampleCount": 3.0,
            "Timestamp": "2016-10-20T12:00:00Z",
            "Average": 3.6666666666666665,
            "Maximum": 5.0,
            "Minimum": 2.0,
            "Sum": 11.0,
            "Unit": "None"
        }
```

```
    ],
    "Label": "PageViewCount"
}
```

Publish statistic sets

You can aggregate your data before you publish to CloudWatch. When you have multiple data points per minute, aggregating data minimizes the number of calls to **put-metric-data**. For example, instead of calling **put-metric-data** multiple times for 3 data points that are within 3 seconds of each other, you can aggregate the data into a statistic set that you publish with one call, using the `--statistic-values` parameter.

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService
 --statistic-values Sum=11,Minimum=2,Maximum=5,SampleCount=3 --
timestamp 2016-10-14T12:00:00.000Z
```

CloudWatch needs raw data points to calculate percentiles. If you publish data using a statistic set instead, you can't retrieve percentile statistics for this data unless one of the following conditions is true:

- The `SampleCount` of the statistic set is 1
- The `Minimum` and the `Maximum` of the statistic set are equal

Publish the value zero

When your data is more sporadic and you have periods that have no associated data, you can choose to publish the value zero (`0`) for that period or no value at all. If you use periodic calls to `PutMetricData` to monitor the health of your application, you might want to publish zero instead of no value. For example, you can set a CloudWatch alarm to notify you if your application fails to publish metrics every five minutes. You want such an application to publish zeros for periods with no associated data.

You might also publish zeros if you want to track the total number of data points or if you want statistics such as minimum and average to include data points with the value 0.

Stop publishing metrics

To stop publishing custom metrics to CloudWatch, change your application's or service's code to stop using **PutMetricData**. CloudWatch doesn't pull metrics from applications, it only receives what is pushed to it, so to stop publishing your metrics you must stop them at the source.

Query your CloudWatch metrics with CloudWatch Metrics Insights

CloudWatch Metrics Insights is a powerful high-performance SQL query engine that you can use to query your metrics at scale. You can identify trends and patterns within all of your CloudWatch metrics in real time, with access to up to two weeks of historical data for comprehensive trend analysis.

You can also set alarms on any Metrics Insights queries that return a single time series. This can be especially useful to create alarms that watch aggregated metrics across a fleet of your infrastructure or applications. Create the alarm once, and it dynamically adjusts as resources are added to or removed from the fleet.

You can perform a CloudWatch Metrics Insights query in the console with the CloudWatch Metrics Insights query editor. You can also perform a CloudWatch Metrics Insights query with the AWS CLI or an AWS SDK by running `GetMetricData` or `PutDashboard`. There's no charge for queries that you run with the CloudWatch Metrics Insights query editor. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

With the CloudWatch Metrics Insights query editor, you can choose from a variety of prebuilt sample queries and also create your own queries. As you create your queries, you can use a builder view to browse your existing metrics and dimensions. Alternatively, use an editor view to manually write queries.

With Metrics Insights, you can run queries at scale. With the **GROUP BY** clause, you can group your metrics in real time into separate time series per specific dimension value. Because Metrics Insights queries include an **ORDER BY** ability, you can use Metrics Insights to make "Top N" type queries. For example, "Top N" type queries can scan millions of metrics in your account and return the 10 instances that consume the most CPU. This can help you pinpoint and remedy latency issues in your applications. To use tags with alarms, opt in through CloudWatch Settings. After tags is enabled, you can also filter and group metrics using AWS resource tags, enabling you to query metrics aligned with your organizational structure, such as by application, environment, or team.

Topics

- Building your queries in CloudWatch Metrics Insights
- Query components and syntax in CloudWatch Metrics Insights
- Alarms on CloudWatch Metrics Insights queries in CloudWatch
- Use Metrics Insights queries with metric math
- Use natural language to generate and update CloudWatch Metrics Insights queries
- SQL inference
- Metrics Insights quotas
- Metrics Insights sample queries
- Metrics Insights glossary
- Troubleshooting Metrics Insights

Building your queries in CloudWatch Metrics Insights

You can run a CloudWatch Metrics Insights query using the CloudWatch console, the AWS CLI, or the AWS SDKs. Queries run in the console are free of charge. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

With CloudWatch Metrics Insights, you can analyze metric data across extended time periods of up to two weeks, allowing for more comprehensive historical analysis and trend identification compared to shorter retention periods. For optimal performance when querying longer time ranges, consider using larger periods (such as 5 minutes or 1 hour) to reduce the number of data points returned. When analyzing trends over the full two-week period, use aggregate functions like AVG() or MAX() in your ORDER BY clauses to efficiently identify patterns.

For more information about using the AWS SDKs to perform a Metrics Insights query, see GetMetricData.

To run a query using the CloudWatch console, follow these steps:

To query your metrics using Metrics Insights

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. If you are satisfied with this query, you can skip the rest of this procedure. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.

4. To create your own query, choose **Multi source query**. You can then use the **Builder** view (default) to get a guided experience, or the **Editor** view if you prefer to see the query syntax. You can switch between the two views anytime and see your work in progress in both views.

   In the **Builder** view, click on the namespace, metric name, filter, group, order, and limit fields to browse and select possible values. You can start typing any part of the value you are looking for to filter the list presented by the builder. You can reference resource tags in the filter and group inputs.

   In the **Editor** view, you can write the query using the subset of SQL supported by Metrics Insights. The editor offers auto-complete options based on the characters you have typed so far, including name of resource tags for metrics that support them.

   CloudWatch Metrics Insights supports querying metrics by AWS resource tags. You can use tags to filter and group your metric data for more targeted monitoring and analysis.

   The following examples shows how you can use queries with tags.

   To see the CPU utilization for Amazon EC2 instances within your production environment:

   ```
   SELECT MAX(CPUUtilization) FROM SCHEMA("AWS/EC2") WHERE tag.env='prod'
   ```

   To group the metrics by environment using the GROUP BY clause:

   ```
   SELECT MAX(CPUUtilization) FROM SCHEMA("AWS/EC2") GROUP BY tag.env
   ```

   To use the GROUP BY clause where you specify the tag name:

   ```
   SELECT AVG(CPUUtilization) FROM "AWS/EC2" GROUP BY tag."aws:cloudformation:stack-
   name"
   ```

   To combine tag queries with existing metric dimensions:

   ```
   SELECT MAX(CPUUtilization) FROM SCHEMA("AWS/EC2") WHERE tag.env='prod' AND
    InstanceId='i-1234567890abcdef0'
   ```

5. When you are satisfied with your query, choose **Run**.

6. (Optional) Another way to edit a query that you have graphed is to choose the **Graphed metrics** tab and choose the edit icon next to the query formula in the **Details** column.

7. (Optional) To remove a query from the graph, choose **Graphed metrics** and choose the **X** icon at the right side of the row that displays your query.

Query components and syntax in CloudWatch Metrics Insights

CloudWatch Metrics Insights syntax is as follows.

```
SELECT FUNCTION(metricName)
FROM namespace | SCHEMA(...)
[ WHERE labelKey OPERATOR labelValue [AND ... ] ]
[ GROUP BY labelKey [ , ... ] ]
[ ORDER BY FUNCTION() [ DESC | ASC ] ]
[ LIMIT number ]
```

The possible clauses in a Metrics Insights query are as follows. None of the keywords are case sensitive, but the identifiers such as the names of metrics, namespaces, and dimensions are case sensitive.

SELECT

Required. Specifies the function to use to aggregate observations in each time bucket (determined by the provided period). Also specifies the name of the metric to query.

The valid values for **FUNCTION** are `AVG`, `COUNT`, `MAX`, `MIN`, and `SUM`.

- `AVG` calculates the average of the observations matched by the query.
- `COUNT` returns the count of the observations matched by the query.
- `MAX` returns the maximum value of the observations matched by the query.
- `MIN` returns the minimum value of the observations matched by the query.
- `SUM` calculates the sum of the observations matched by the query.

FROM

Required. Specifies the source of the metric. You can specify either the metric namespace that contains the metric that is to be queried, or a **SCHEMA** table function. Examples of metric namespaces include `"AWS/EC2"`, `"AWS/Lambda"`, and metric namespaces that you have created for your custom metrics.

Metric namespaces that include **/** or any other character that is not a letter, number, or underscore must be surrounded by double quotation marks. For more information, see What needs quotation marks or escape characters?.

SCHEMA

An optional table function that can be used within a **FROM** clause. Use **SCHEMA** to scope down the query results to only the metrics that exactly match a list of dimensions, or to metrics that have no dimensions.

If you use a **SCHEMA** clause, it must contain at least one argument, and this first argument must be the metric namespace being queried. If you specify **SCHEMA** with only this namespace argument, the results are scoped down to only metrics that do not have any dimensions.

If you specify **SCHEMA** with additional arguments, the additional arguments after the namespace argument must be *label* keys. Label keys must be dimension names. If you specify one or more of these label keys, the results are scoped down to only those metrics that have that exact set of dimensions. The order of these label keys does not matter.

For example:

- **SELECT AVG(CPUUtilization) FROM "AWS/EC2"** matches all `CPUUtilization` metrics in the `AWS/EC2` namespace, no matter their dimensions, and returns a single aggregated time series.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2")** matches only the `CPUUtilization` metrics in the `AWS/EC2` namespace that do not have any dimensions defined.
- **SELECT AVG(CPUUtilization) FROM SCHEMA("AWS/EC2", InstanceId)** matches only the `CPUUtilization` metrics that were reported to CloudWatch with exactly one dimension, `InstanceId`.
- **SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)** matches only the `RequestCount` metrics that were reported to CloudWatch from `AWS/ApplicationELB` with exactly two dimensions, `LoadBalancer` and `AvailabilityZone`.

WHERE

Optional. Filters the results to only those metrics that match your specified expression using specific label values for one or more label keys. For example, **WHERE InstanceType =**

'c3.4xlarge' filters the results to only `c3.4xlarge` instance types, and **WHERE InstanceType != 'c3.4xlarge'** filters the results to all instance types except `c3.4xlarge`.

When you run a query in a monitoring account, you can use `WHERE AWS.AccountId` to limit results to only the account that you specify. For example, `WHERE AWS.AccountId=444455556666` queries metrics from only account `444455556666`. To limit your query to only metrics in the monitoring account itself, use `WHERE AWS.AccountId=CURRENT_ACCOUNT_ID()`.

Label values must always be enclosed with single quotation marks.

Using tags in WHERE clauses

You can filter results by AWS resource tags using the syntax `tag.keyName`. Tag filters follow the same operator rules as dimension filters. For example:

- WHERE `tag.env = 'prod'` filters to metrics from resources tagged with *env=prod*
- WHERE `tag.department != 'test'` excludes metrics from resources tagged with *department=test*

Tag filters can be combined with dimension filters:

```
WHERE tag.env = 'prod' AND InstanceType = 'm5.large'
```

Supported operators

The **WHERE** clause supports the following operators:

- **=** Label value must match the specified string.
- **!=** Label value must not match the specified string.
- **AND** Both conditions that are specified must be true to match. You can use multiple **AND** keywords to specify two or more conditions.

GROUP BY

Optional. Groups the query results into multiple time series, each one corresponding to a different value for the specified label key or keys. For example, using `GROUP BY InstanceId` returns a different time series for each value of `InstanceId`. Using `GROUP BY ServiceName, Operation` creates a different time series for each possible combination of the values of `ServiceName` and `Operation`.

With a **GROUP BY** clause, by default the results are ordered in alphabetical ascending order, using the sequence of labels specified in the **GROUP BY** clause. To change the order of the results, add an **ORDER BY** clause to your query.

When you run a query in a monitoring account, you can use `GROUP BY AWS.AccountId` to group the results based on the accounts they are from.

Using tags in GROUP BY clauses

You can group results by AWS resource tag values using the syntax `tag.keyName`. For example:

- *GROUP BY tag.environment* creates separate time series for each environment tag value
- *GROUP BY tag.team, InstanceType* groups by both tag and dimension values
- *GROUP BY tag.team, AWS.AccountId* groups by both tag and linked source AccountIDs

> **Note**
>
> If some of the matching metrics don't include a specific label key specified in the **GROUP BY** clause, a null group named `Other` is returned. For example, if you specify `GROUP BY ServiceName, Operation` and some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

ORDER BY

Optional. Specifies the order to use for the returned time series, if the query returns more than one time series. The order is based on the values found by the **FUNCTION** that you specify in the **ORDER BY** clause. The **FUNCTION** is used to calculate a single scalar value from each returned time series, and that value is used to determine the order.

You also specify whether to use ascending **ASC** or descending **DESC** order. If you omit this, the default is ascending **ASC**.

For example, adding an `ORDER BY MAX() DESC` clause orders the results by the maximum data point observed within the time range, in descending order: meaning that the time series that has the highest maximum data point is returned first.

The valid functions to use within an **ORDER BY** clause are `AVG()`, `COUNT()`, `MAX()`, `MIN()`, and `SUM()`.

If you use an **ORDER BY** clause with a **LIMIT** clause, the resulting query is a "Top N" query. **ORDER BY** is also useful for queries that might return a large number of metrics, because each query can return no more than 500 time series. If a query matches more than 500 time series, and you use an **ORDER BY** clause, the time series are sorted and then the 500 time series that come first in the sort order are the ones that are returned.

LIMIT

Optional. Limits the number of time series returned by the query to the value that you specify. The maximum value that you can specify is 500, and a query that does not specify a **LIMIT** can also return no more than 500 time series.

Using a **LIMIT** clause with an **ORDER BY** clause gives you a "Top N" query.

What needs quotation marks or escape characters?

In a query, label values must always be surrounded with single quotation marks. For example, **SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE AutoScalingGroupName = 'my-production-fleet'**.

Metric namespaces, metric names, and label keys that contain characters other than letters, numbers, and underscore (_) must be surrounded by double quote marks. For example, **SELECT MAX("My.Metric")**.

If one of these contains a double quotation mark or single quotation mark itself (such as `Bytes"Input"`), you must escape each quotation mark with a backslash, as in **SELECT AVG("Bytes \"Input\"")**.

If a metric namespace, metric name, or label key, contains a word that is a reserved keyword in Metrics Insights, these must also be enclosed in double quotation marks. For example, if you have a metric named `LIMIT`, you would use `SELECT AVG("LIMIT")`. It is also valid to enclose any namespace, metric name, or label in double quotation marks even if it does not include a reserved keyword.

For a complete list of reserved keywords, see Reserved keywords.

Build a rich query step by step

This section illustrates building a full example that uses all possible clauses, step by step.

You can start with the following query, which aggregates all of the Application Load Balancer `RequestCount` metrics that are collected with both the dimensions `LoadBalancer` and `AvailabilityZone`.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
```

To see metrics only from a specific load balancer, you can add a **WHERE** clause to limit the metrics returned to only those metrics where the value of the `LoadBalancer` dimension is `app/load-balancer-1`.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
```

The preceding query aggregates the `RequestCount` metrics from all Availability Zones for this load balancer into one time series. If you want to see different time series for each Availability Zone, we can add a **GROUP BY** clause.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
```

Next, you can order the results to see the highest values first. The following **ORDER BY** clause orders the time series in descending order, by the maximum value reported by each time series during the query time range:

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
```

You can also use tags to further filter the results. For example, if you want to see results only for load balancers tagged with a specific environment, you can add tag filtering to the WHERE clause:

```
SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer,
 AvailabilityZone) WHERE LoadBalancer = 'app/load-balancer-1' AND tag.Environment =
 'prod' GROUP BY AvailabilityZone ORDER BY MAX() DESC
```

You can also group the results by tag values instead of (or in addition to) dimensions. For example, grouping by the Application tag:

```
SELECT SUM(RequestCount) FROM SCHEMA("AWS/ApplicationELB", LoadBalancer,
 AvailabilityZone) WHERE tag.Environment = 'prod' GROUP BY tag.Application ORDER BY
 MAX() DESC
```

Finally, if we are primarily interested in a "Top N" type of query, we can use a **LIMIT** clause. This final example limits the results to only the time series with the five highest `MAX` values.

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer, AvailabilityZone)
WHERE LoadBalancer = 'app/load-balancer-1'
GROUP BY AvailabilityZone
ORDER BY MAX() DESC
LIMIT 5
```

Cross-account query examples

These examples are valid when run in an account set up as a monitoring account in CloudWatch cross-account observability.

The following example searches all Amazon EC2 instances in the source account 123456789012 and returns the average.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId ='123456789012'
```

The following example queries the `CPUUtilization` metric in `AWS/EC2` in all the linked source accounts, and groups the results by account ID and instance type.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
GROUP BY AWS.AccountId, InstanceType
```

The following example queries the `CPUUtilization` in the monitoring account itself.

```
SELECT AVG(CpuUtilization)
FROM "AWS/EC2"
WHERE AWS.AccountId = CURRENT_ACCOUNT_ID()
```

Reserved keywords

The following are reserved keywords in CloudWatch Metrics Insights. If any of these words are in a namespace, metric name, or label key in a query, you must enclose them in double quote marks. Reserved keywords are not case sensitive.

```
"ABORT" "ABORTSESSION" "ABS" "ABSOLUTE" "ACCESS" "ACCESSIBLE" "ACCESS_LOCK" "ACCOUNT"
 "ACOS" "ACOSH" "ACTION" "ADD" "ADD_MONTHS"
"ADMIN" "AFTER" "AGGREGATE" "ALIAS" "ALL" "ALLOCATE" "ALLOW" "ALTER" "ALTERAND" "AMP"
 "ANALYSE" "ANALYZE" "AND" "ANSIDATE" "ANY" "ARE" "ARRAY",
"ARRAY_AGG" "ARRAY_EXISTS" "ARRAY_MAX_CARDINALITY" "AS" "ASC" "ASENSITIVE" "ASIN"
 "ASINH" "ASSERTION" "ASSOCIATE" "ASUTIME" "ASYMMETRIC" "AT",
"ATAN" "ATAN2" "ATANH" "ATOMIC" "AUDIT" "AUTHORIZATION" "AUX" "AUXILIARY" "AVE"
 "AVERAGE" "AVG" "BACKUP" "BEFORE" "BEGIN" "BEGIN_FRAME" "BEGIN_PARTITION",
"BETWEEN" "BIGINT" "BINARY" "BIT" "BLOB" "BOOLEAN" "BOTH" "BREADTH" "BREAK" "BROWSE"
 "BT" "BUFFERPOOL" "BULK" "BUT" "BY" "BYTE" "BYTEINT" "BYTES" "CALL",
"CALLED" "CAPTURE" "CARDINALITY" "CASCADE" "CASCADED" "CASE" "CASESPECIFIC" "CASE_N"
 "CAST" "CATALOG" "CCSID" "CD" "CEIL" "CEILING" "CHANGE" "CHAR",
"CHAR2HEXINT" "CHARACTER" "CHARACTERS" "CHARACTER_LENGTH" "CHARS" "CHAR_LENGTH" "CHECK"
 "CHECKPOINT" "CLASS" "CLASSIFIER" "CLOB" "CLONE" "CLOSE" "CLUSTER",
"CLUSTERED" "CM" "COALESCE" "COLLATE" "COLLATION" "COLLECT" "COLLECTION" "COLLID"
 "COLUMN" "COLUMN_VALUE" "COMMENT" "COMMIT" "COMPLETION" "COMPRESS" "COMPUTE",
"CONCAT" "CONCURRENTLY" "CONDITION" "CONNECT" "CONNECTION" "CONSTRAINT" "CONSTRAINTS"
 "CONSTRUCTOR" "CONTAINS" "CONTAINSTABLE" "CONTENT" "CONTINUE" "CONVERT",
"CONVERT_TABLE_HEADER" "COPY" "CORR" "CORRESPONDING" "COS" "COSH" "COUNT" "COVAR_POP"
 "COVAR_SAMP" "CREATE" "CROSS" "CS" "CSUM" "CT" "CUBE" "CUME_DIST",
"CURRENT" "CURRENT_CATALOG" "CURRENT_DATE" "CURRENT_DEFAULT_TRANSFORM_GROUP"
 "CURRENT_LC_CTYPE" "CURRENT_PATH" "CURRENT_ROLE" "CURRENT_ROW" "CURRENT_SCHEMA",
"CURRENT_SERVER" "CURRENT_TIME" "CURRENT_TIMESTAMP" "CURRENT_TIMEZONE"
 "CURRENT_TRANSFORM_GROUP_FOR_TYPE" "CURRENT_USER" "CURRVAL" "CURSOR" "CV" "CYCLE"
 "DATA",
"DATABASE" "DATABASES" "DATABLOCKSIZE" "DATE" "DATEFORM" "DAY" "DAYS" "DAY_HOUR"
 "DAY_MICROSECOND" "DAY_MINUTE" "DAY_SECOND" "DBCC" "DBINFO" "DEALLOCATE" "DEC",
"DECFLOAT" "DECIMAL" "DECLARE" "DEFAULT" "DEFERRABLE" "DEFERRED" "DEFINE" "DEGREES"
 "DEL" "DELAYED" "DELETE" "DENSE_RANK" "DENY" "DEPTH" "DEREF" "DESC" "DESCRIBE",
"DESCRIPTOR" "DESTROY" "DESTRUCTOR" "DETERMINISTIC" "DIAGNOSTIC" "DIAGNOSTICS"
 "DICTIONARY" "DISABLE" "DISABLED" "DISALLOW" "DISCONNECT" "DISK" "DISTINCT",
```

```
"DISTINCTROW" "DISTRIBUTED" "DIV" "DO" "DOCUMENT" "DOMAIN" "DOUBLE" "DROP" "DSSIZE"
 "DUAL" "DUMP" "DYNAMIC" "EACH" "ECHO" "EDITPROC" "ELEMENT" "ELSE" "ELSEIF",
"EMPTY" "ENABLED" "ENCLOSED" "ENCODING" "ENCRYPTION" "END" "END-EXEC" "ENDING"
 "END_FRAME" "END_PARTITION" "EQ" "EQUALS" "ERASE" "ERRLVL" "ERROR" "ERRORFILES",
"ERRORTABLES" "ESCAPE" "ESCAPED" "ET" "EVERY" "EXCEPT" "EXCEPTION" "EXCLUSIVE" "EXEC"
 "EXECUTE" "EXISTS" "EXIT" "EXP" "EXPLAIN" "EXTERNAL" "EXTRACT" "FALLBACK
"FALSE" "FASTEXPORT" "FENCED" "FETCH" "FIELDPROC" "FILE" "FILLFACTOR" "FILTER" "FINAL"
 "FIRST" "FIRST_VALUE" "FLOAT" "FLOAT4" "FLOAT8" "FLOOR"
"FOR" "FORCE" "FOREIGN" "FORMAT" "FOUND" "FRAME_ROW" "FREE" "FREESPACE" "FREETEXT"
 "FREETEXTTABLE" "FREEZE" "FROM" "FULL" "FULLTEXT" "FUNCTION"
"FUSION" "GE" "GENERAL" "GENERATED" "GET" "GIVE" "GLOBAL" "GO" "GOTO" "GRANT" "GRAPHIC"
 "GROUP" "GROUPING" "GROUPS" "GT" "HANDLER" "HASH"
"HASHAMP" "HASHBAKAMP" "HASHBUCKET" "HASHROW" "HAVING" "HELP" "HIGH_PRIORITY" "HOLD"
 "HOLDLOCK" "HOUR" "HOURS" "HOUR_MICROSECOND" "HOUR_MINUTE"
"HOUR_SECOND" "IDENTIFIED" "IDENTITY" "IDENTITYCOL" "IDENTITY_INSERT" "IF" "IGNORE"
 "ILIKE" "IMMEDIATE" "IN" "INCLUSIVE" "INCONSISTENT" "INCREMENT"
"INDEX" "INDICATOR" "INFILE" "INHERIT" "INITIAL" "INITIALIZE" "INITIALLY" "INITIATE"
 "INNER" "INOUT" "INPUT" "INS" "INSENSITIVE" "INSERT" "INSTEAD"
"INT" "INT1" "INT2" "INT3" "INT4" "INT8" "INTEGER" "INTEGERDATE" "INTERSECT"
 "INTERSECTION" "INTERVAL" "INTO" "IO_AFTER_GTIDS" "IO_BEFORE_GTIDS"
"IS" "ISNULL" "ISOBID" "ISOLATION" "ITERATE" "JAR" "JOIN" "JOURNAL" "JSON_ARRAY"
 "JSON_ARRAYAGG" "JSON_EXISTS" "JSON_OBJECT" "JSON_OBJECTAGG"
"JSON_QUERY" "JSON_TABLE" "JSON_TABLE_PRIMITIVE" "JSON_VALUE" "KEEP" "KEY" "KEYS"
 "KILL" "KURTOSIS" "LABEL" "LAG" "LANGUAGE" "LARGE" "LAST"
"LAST_VALUE" "LATERAL" "LC_CTYPE" "LE" "LEAD" "LEADING" "LEAVE" "LEFT" "LESS" "LEVEL"
 "LIKE" "LIKE_REGEX" "LIMIT" "LINEAR" "LINENO" "LINES"
"LISTAGG" "LN" "LOAD" "LOADING" "LOCAL" "LOCALE" "LOCALTIME" "LOCALTIMESTAMP" "LOCATOR"
 "LOCATORS" "LOCK" "LOCKING" "LOCKMAX" "LOCKSIZE" "LOG"
"LOG10" "LOGGING" "LOGON" "LONG" "LONGBLOB" "LONGTEXT" "LOOP" "LOWER" "LOW_PRIORITY"
 "LT" "MACRO" "MAINTAINED" "MAP" "MASTER_BIND"
"MASTER_SSL_VERIFY_SERVER_CERT" "MATCH" "MATCHES" "MATCH_NUMBER" "MATCH_RECOGNIZE"
 "MATERIALIZED" "MAVG" "MAX" "MAXEXTENTS" "MAXIMUM" "MAXVALUE"
"MCHARACTERS" "MDIFF" "MEDIUMBLOB" "MEDIUMINT" "MEDIUMTEXT" "MEMBER" "MERGE" "METHOD"
 "MICROSECOND" "MICROSECONDS" "MIDDLEINT" "MIN" "MINDEX"
"MINIMUM" "MINUS" "MINUTE" "MINUTES" "MINUTE_MICROSECOND" "MINUTE_SECOND" "MLINREG"
 "MLOAD" "MLSLABEL" "MOD" "MODE" "MODIFIES" "MODIFY"
"MODULE" "MONITOR" "MONRESOURCE" "MONSESSION" "MONTH" "MONTHS" "MSUBSTR" "MSUM"
 "MULTISET" "NAMED" "NAMES" "NATIONAL" "NATURAL" "NCHAR" "NCLOB"
"NE" "NESTED_TABLE_ID" "NEW" "NEW_TABLE" "NEXT" "NEXTVAL" "NO" "NOAUDIT" "NOCHECK"
 "NOCOMPRESS" "NONCLUSTERED" "NONE" "NORMALIZE" "NOT" "NOTNULL"
"NOWAIT" "NO_WRITE_TO_BINLOG" "NTH_VALUE" "NTILE" "NULL" "NULLIF" "NULLIFZERO" "NULLS"
 "NUMBER" "NUMERIC" "NUMPARTS" "OBID" "OBJECT" "OBJECTS"
"OCCURRENCES_REGEX" "OCTET_LENGTH" "OF" "OFF" "OFFLINE" "OFFSET" "OFFSETS" "OLD"
 "OLD_TABLE" "OMIT" "ON" "ONE" "ONLINE" "ONLY" "OPEN" "OPENDATASOURCE"
```

```
"OPENQUERY" "OPENROWSET" "OPENXML" "OPERATION" "OPTIMIZATION" "OPTIMIZE"
 "OPTIMIZER_COSTS" "OPTION" "OPTIONALLY" "OR" "ORDER" "ORDINALITY" "ORGANIZATION"
"OUT" "OUTER" "OUTFILE" "OUTPUT" "OVER" "OVERLAPS" "OVERLAY" "OVERRIDE" "PACKAGE" "PAD"
 "PADDED" "PARAMETER" "PARAMETERS" "PART" "PARTIAL" "PARTITION"
"PARTITIONED" "PARTITIONING" "PASSWORD" "PATH" "PATTERN" "PCTFREE" "PER" "PERCENT"
 "PERCENTILE" "PERCENTILE_CONT" "PERCENTILE_DISC" "PERCENT_RANK" "PERIOD" "PERM"
"PERMANENT" "PIECESIZE" "PIVOT" "PLACING" "PLAN" "PORTION" "POSITION" "POSITION_REGEX"
 "POSTFIX" "POWER" "PRECEDES" "PRECISION" "PREFIX" "PREORDER"
"PREPARE" "PRESERVE" "PREVVAL" "PRIMARY" "PRINT" "PRIOR" "PRIQTY" "PRIVATE"
 "PRIVILEGES" "PROC" "PROCEDURE" "PROFILE" "PROGRAM" "PROPORTIONAL"
"PROTECTION" "PSID" "PTF" "PUBLIC" "PURGE" "QUALIFIED" "QUALIFY" "QUANTILE" "QUERY"
 "QUERYNO" "RADIANS" "RAISERROR" "RANDOM" "RANGE" "RANGE_N" "RANK"
"RAW" "READ" "READS" "READTEXT" "READ_WRITE" "REAL" "RECONFIGURE" "RECURSIVE" "REF"
 "REFERENCES" "REFERENCING" "REFRESH" "REGEXP" "REGR_AVGX" "REGR_AVGY"
"REGR_COUNT" "REGR_INTERCEPT" "REGR_R2" "REGR_SLOPE" "REGR_SXX" "REGR_SXY" "REGR_SYY"
 "RELATIVE" "RELEASE" "RENAME" "REPEAT" "REPLACE" "REPLICATION"
"REPOVERRIDE" "REQUEST" "REQUIRE" "RESIGNAL" "RESOURCE" "RESTART" "RESTORE" "RESTRICT"
 "RESULT" "RESULT_SET_LOCATOR" "RESUME" "RET" "RETRIEVE" "RETURN"
"RETURNING" "RETURNS" "REVALIDATE" "REVERT" "REVOKE" "RIGHT" "RIGHTS" "RLIKE" "ROLE"
 "ROLLBACK" "ROLLFORWARD" "ROLLUP" "ROUND_CEILING" "ROUND_DOWN"
"ROUND_FLOOR" "ROUND_HALF_DOWN" "ROUND_HALF_EVEN" "ROUND_HALF_UP" "ROUND_UP" "ROUTINE"
 "ROW" "ROWCOUNT" "ROWGUIDCOL" "ROWID" "ROWNUM" "ROWS" "ROWSET"
"ROW_NUMBER" "RULE" "RUN" "RUNNING" "SAMPLE" "SAMPLEID" "SAVE" "SAVEPOINT" "SCHEMA"
 "SCHEMAS" "SCOPE" "SCRATCHPAD" "SCROLL" "SEARCH" "SECOND" "SECONDS"
"SECOND_MICROSECOND" "SECQTY" "SECTION" "SECURITY" "SECURITYAUDIT" "SEEK" "SEL"
 "SELECT" "SEMANTICKEYPHRASETABLE" "SEMANTICSIMILARITYDETAILSTABLE"
"SEMANTICSIMILARITYTABLE" "SENSITIVE" "SEPARATOR" "SEQUENCE" "SESSION" "SESSION_USER"
 "SET" "SETRESRATE" "SETS" "SETSESSRATE" "SETUSER" "SHARE" "SHOW"
"SHUTDOWN" "SIGNAL" "SIMILAR" "SIMPLE" "SIN" "SINH" "SIZE" "SKEW" "SKIP" "SMALLINT"
 "SOME" "SOUNDEX" "SOURCE" "SPACE" "SPATIAL" "SPECIFIC" "SPECIFICTYPE"
"SPOOL" "SQL" "SQLEXCEPTION" "SQLSTATE" "SQLTEXT" "SQLWARNING" "SQL_BIG_RESULT"
 "SQL_CALC_FOUND_ROWS" "SQL_SMALL_RESULT" "SQRT" "SS" "SSL" "STANDARD"
"START" "STARTING" "STARTUP" "STAT" "STATE" "STATEMENT" "STATIC" "STATISTICS" "STAY"
 "STDDEV_POP" "STDDEV_SAMP" "STEPINFO" "STOGROUP" "STORED" "STORES"
"STRAIGHT_JOIN" "STRING_CS" "STRUCTURE" "STYLE" "SUBMULTISET" "SUBSCRIBER" "SUBSET"
 "SUBSTR" "SUBSTRING" "SUBSTRING_REGEX" "SUCCEEDS" "SUCCESSFUL"
"SUM" "SUMMARY" "SUSPEND" "SYMMETRIC" "SYNONYM" "SYSDATE" "SYSTEM" "SYSTEM_TIME"
 "SYSTEM_USER" "SYSTIMESTAMP" "TABLE" "TABLESAMPLE" "TABLESPACE" "TAN"
"TANH" "TBL_CS" "TEMPORARY" "TERMINATE" "TERMINATED" "TEXTSIZE" "THAN" "THEN"
 "THRESHOLD" "TIME" "TIMESTAMP" "TIMEZONE_HOUR" "TIMEZONE_MINUTE" "TINYBLOB"
"TINYINT" "TINYTEXT" "TITLE" "TO" "TOP" "TRACE" "TRAILING" "TRAN" "TRANSACTION"
 "TRANSLATE" "TRANSLATE_CHK" "TRANSLATE_REGEX" "TRANSLATION" "TREAT"
"TRIGGER" "TRIM" "TRIM_ARRAY" "TRUE" "TRUNCATE" "TRY_CONVERT" "TSEQUAL" "TYPE" "UC"
 "UESCAPE" "UID" "UNDEFINED" "UNDER" "UNDO" "UNION" "UNIQUE"
```

```
"UNKNOWN" "UNLOCK" "UNNEST" "UNPIVOT" "UNSIGNED" "UNTIL" "UPD" "UPDATE" "UPDATETEXT"
 "UPPER" "UPPERCASE" "USAGE" "USE" "USER" "USING" "UTC_DATE"
"UTC_TIME" "UTC_TIMESTAMP" "VALIDATE" "VALIDPROC" "VALUE" "VALUES" "VALUE_OF"
 "VARBINARY" "VARBYTE" "VARCHAR" "VARCHAR2" "VARCHARACTER" "VARGRAPHIC"
"VARIABLE" "VARIADIC" "VARIANT" "VARYING" "VAR_POP" "VAR_SAMP" "VCAT" "VERBOSE"
 "VERSIONING" "VIEW" "VIRTUAL" "VOLATILE" "VOLUMES" "WAIT" "WAITFOR"
"WHEN" "WHENEVER" "WHERE" "WHILE" "WIDTH_BUCKET" "WINDOW" "WITH" "WITHIN"
 "WITHIN_GROUP" "WITHOUT" "WLM" "WORK" "WRITE" "WRITETEXT" "XMLCAST" "XMLEXISTS"
"XMLNAMESPACES" "XOR" "YEAR" "YEARS" "YEAR_MONTH" "ZEROFILL" "ZEROIFNULL" "ZONE"
```

Alarms on CloudWatch Metrics Insights queries in CloudWatch

You can create alarms on Metrics Insights queries. This helps you have alarms that track multiple resources without needing to be updated later. The query catches new resources and resources that change. For example, you can create an alarm that watches the CPU utilization of your fleet, and the alarm automatically evaluates new instances that you launch after creating the alarm.

In a monitoring account that is set up for CloudWatch cross-account observability, your Metrics Insights alarms can watch resources in source accounts and in the monitoring account itself. For more information about how to limit your alarm queries to a specific account or to group the results by account ID, see the `WHERE` and `GROUP BY` sections in Query components and syntax in CloudWatch Metrics Insights.

Using tags in alarm queries

You can create alarms on Metrics Insights queries that use AWS resource tags to filter and group metrics. To use tags with alarms, on the CloudWatch console, choose **Settings**. On the **CloudWatch Settings** page, under **Enable resource tags on telemetry**, choose **Enable**. Context-aware alarms monitor specific applications, environments, or teams automatically as resources change.

For example, you can create an alarm that monitors CPU utilization for all Amazon EC2 instances tagged with a specific application.

```
SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE tag.Application = 'Orders' AND
 tag.Environment = 'Prod'
```

Tag-based alarms automatically adapt as you add or remove resources with matching tags, providing dynamic monitoring aligned with your operational structure.

Contents

- Creating a Metrics Insights CloudWatch alarm

Creating a Metrics Insights CloudWatch alarm

To create an alarm on a Metrics Insights query using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Classic metrics**.
3. (Optional) To run a pre-built sample query, choose **Add query** and select the query to run. Or, you can choose **Editor** to edit the sample query and then choose **Run** to run the modified query.
4. To create your own query, choose **Multi source query**. You can then use the **Builder** view, the **Editor** view, and also use a combination of both. You can switch between the two views anytime and see your work in progress in both views.

   In the **Builder** view, you can browse and select the metric namespace, metric name, filter, group, and order options. For each of these options, the query builder offers you a list of possible selections from your environment to choose from.

   In the **Editor** view, you can start writing your query. As you type, the editor offers suggestions based on the characters that you have typed so far.

   For example, when creating your Metrics Insights query for the alarm, you can use tags to filter and group metrics for more targeted monitoring.

   - Filter by tags – Use `WHERE tag.keyName = 'value'` to monitor resources with specific tags

     ```
     SELECT MAX(CPUUtilization) FROM "AWS/EC2" WHERE tag.Environment = 'Prod'
     ```

   - Combine tags with dimensions – Mix tag filters with existing metric dimensions

     ```
     SELECT AVG(Duration) FROM "AWS/Lambda" WHERE tag.Application = 'OrderService' AND
      FunctionName = 'process%'
     ```

   > **Note**
   >
   > When using tags, alarms will match the metrics only if the specified tags existed on the associated resources during the evaluated time period.

5. When you are satisfied with your query, choose **Run**.
6. Choose **Create alarm**.
7. Under **Conditions**, specify the following:

   a. For **Whenever *metric* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

   b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

      To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.

   c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
8. Choose **Next**.
9. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

   To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

   To have the alarm not send notifications, choose **Remove**.
10. To have the alarm perform Auto Scaling, EC2, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

11. When finished, choose **Next**.

12. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
13. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

To create an alarm on a Metrics Insights query using the AWS CLI

Use the `put-metric-alarm` command and specify a Metrics Insights query in the `metrics` parameter. For example, the following command sets an alarm that goes into ALARM state if any of your instances go above 80% in CPU utilization.

```
aws cloudwatch put-metric-alarm --alarm-name Prod-App-CPU-Alarm --evaluation-periods 1
 --comparison-operator GreaterThanThreshold --metrics '[{"Id":"m1","Expression":"SELECT
 MAX(CPUUtilization) FROM \"AWS/EC2\" WHERE tag.Environment = '\''Prod'\'' AND
 tag.Application = '\''OrderService'\''", "Period":60}]' --threshold 80
```

Use Metrics Insights queries with metric math

You can use a Metrics Insights query as an input to a metric math function. For more information about metric math, see Math expressions with metrics.

A Metrics Insights query that does not include a **GROUP BY** clause returns a single time series. Therefore, its returned results can be used with any metric math function that takes a single time series as input.

A Metrics Insights query that includes a **GROUP BY** clause returns multiple time series. Therefore, its returned results can be used with any metric math function that takes an array of time series as input.

For example, the following query returns the total number of bytes downloaded for each bucket in the Region, as an array of time series:

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
```

On a graph in the console or in a GetMetricData operation, the results of this query are q1. This query returns the result in bytes, so if you want to see the result as MB instead, you can use the following math function:

```
q1/1024/1024
```

Use natural language to generate and update CloudWatch Metrics Insights queries

CloudWatch supports a natural language query capability to help you generate and update queries for CloudWatch Metrics Insights and CloudWatch Logs Insights.

With this capability, you can ask questions about or describe the CloudWatch data you're looking for in plain English. The natural language capability generates a query based on a prompt that you enter and provides a line-by-line explanation of how the query works. You can also update your query to further investigate your data.

Depending on your environment, you can enter prompts like "Which Amazon Elastic Compute Cloud instance has the highest network out?" and "Show me the top 10 Amazon DynamoDB Tables by consumed reads."

Note

The natural-language query feature is generally available in 10 Regions. For some Regions, the feature makes cross-Region calls to Regions in the United States to process the query prompts. The following table lists the supported Regions, and shows where each Region processes its prompts.

Supported Region	Region where prompt is processed
US East (N. Virginia)	US East (N. Virginia)
US East (Ohio)	US East (N. Virginia)
US West (Oregon)	US West (Oregon)
Asia Pacific (Hong Kong)	US West (Oregon)
Asia Pacific (Singapore)	US West (Oregon)
Asia Pacific (Sydney)	US West (Oregon)

Supported Region	Region where prompt is processed
Asia Pacific (Tokyo)	Asia Pacific (Tokyo)
Europe (Frankfurt)	Europe (Frankfurt)
Europe (Ireland)	US East (N. Virginia)
Europe (Stockholm)	US East (N. Virginia)

To generate a CloudWatch Metrics Insights query with this capability, open the CloudWatch Metrics Insights query editor in the *builder* or *editor* view and choose **Generate query**.

Important

To use the natural language query capability, you must use the CloudWatchFullAccess, CloudWatchReadOnlyAccess, CloudWatchFullAccessV2, AdministratorAccess, or ReadOnlyAccess policy.
You can also include the `cloudwatch:GenerateQuery` action in a new or existing customer managed or inline policy.

Example queries

The examples in this section describe how to generate and update queries using the natural language capability.

Note

For more information on the CloudWatch Metrics Insights query editor and syntax, see CloudWatch Metrics Insights query components and syntax.

Example: Generate a natural language query

To generate a query using natural language, enter a prompt and choose **Generate new query**. This example shows a query that performs a basic search.

Prompt

The following is an example of a prompt that directs the capability to search for the top 10 DynamoDB Tables that consume the most read capacity.

```
Show top 10 DynamoDB Tables by consumed reads
```

Query

The following is an example of a query that the natural language capability generates based on the prompt. Notice how the prompt appears in a comment before the query. After the query, you can read an explanation that describes how the query works.

```
# Show top 10 DynamoDB Tables by consumed reads
SELECT SUM("ConsumedReadCapacityUnits")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query selects the sum of consumed read capacity units for each DynamoDB table,
 groups the results by table name, orders the results from highest to lowest read
 capacity consumption, and limits the results to the top 10 tables.
```

> **Note**
>
> To turn off the appearance of your prompt and the explanation of how the query works, use the gear icon in your editor.

Example: Update a natural language query

You can update a query by editing the initial prompt and then choosing **Update query**.

Updated prompt

The following example shows an updated version of the previous prompt. Instead of a prompt that searches for the top 10 DynamoDB Tables that consume the most read capacity, this prompt now directs the capability to sort the results by the number of bytes returned.

```
Sort by bytes returned instead
```

Updated query

The following is an example of the updated query. Notice how the updated prompt appears in a comment before the updated query. After the query, you can read an explanation that describes how the original query has been updated.

```
# Sort by bytes returned instead
SELECT SUM("ReturnedBytes")
FROM "AWS/DynamoDB"
GROUP BY TableName
ORDER BY SUM() DESC
LIMIT 10
# This query modifies the original query to select the sum of returned bytes instead
 of consumed read capacity units, and orders the results from highest to lowest sum of
 returned bytes, limiting the results to the top 10 tables.
```

Opting out of using your data for service improvement

The natural language prompt data you provide to train the AI model and generate relevant queries is used solely to provide and maintain your service. This data might be used to improve the quality of CloudWatch Metrics Insights. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see AWS Service Terms and AWS responsible AI policy.

You can opt out of having your content used to develop or improve the quality of natural language queries by creating an AI service opt-out policy. To opt-out of data collection for all CloudWatch AI features, including the query generation capability, you must create an opt-out policy for CloudWatch. For more information, see AI services opt-out policies in the *AWS Organizations User Guide*.

SQL inference

CloudWatch Metrics Insights uses several mechanisms to infer the intention of a given SQL query.

Topics

- Time bucketing
- Fields projection
- ORDER BY global aggregation

Time bucketing

Time series data points resulting from a query are rolled up into time buckets based on the requested period. To aggregate values in standard SQL, an explicit GROUP BY clause must be

defined to collect all the observations of a given period together. Because this is the standard way to query time series data, CloudWatch Metrics Insights infers time bucketing without the need to express an explicit **GROUP BY** clause.

For example, when a query is performed with a period of one minute, all the observations belonging to that minute until the next (excluded) are rolled up to the start time of the time bucket. This makes Metrics Insights SQL statements more concise and less verbose.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

The previous query returns a single time series (timestamp-value pairs), representing the average CPU utilization of all Amazon EC2 instances. Assuming the requested period is one minute, each data point returned represents the average of all observations measured within a specific one-minute interval (start time inclusive, end time exclusive). The timestamp related to the specific data point is the start time of the bucket

Fields projection

Metrics Insights queries always return the timestamp projection. You don't need to specify a timestamp column in the **SELECT** clause to get the timestamp of each corresponding data point value. For details about how timestamp is calculated, see Time bucketing.

When using **GROUP BY**, each group name is also inferred and projected in the result, so that you can group the returned time series.

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
```

The previous query returns a time series for each Amazon EC2 instance. Each time series is labeled after the value of the instance ID.

ORDER BY global aggregation

When using **ORDER BY**, **FUNCTION()** infers which aggregate function that you want to order by (the data point values of the queried metrics). The aggregate operation is performed across all the matched data points of each individual time series across the queried time window.

```
SELECT AVG(CPUUtilization)
```

```
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX()
LIMIT 10
```

The previous query returns the CPU utilization for each Amazon EC2 instance, limiting the result set to 10 entries. The results are ordered based on the maximum value of the individual time series within the requested time window. The **ORDER BY** clause is applied before **LIMIT**, so that the ordering is calculated against more than 10 time series.

Metrics Insights quotas

CloudWatch Metrics Insights currently has the following quotas:

- Query up to two weeks of data for visualization in your metrics, widgets, and alarm graphs. You can query the most recent three hours of data for alarm condition evaluations.
- A single query can process no more than 10,000 metrics. This means that if the **SELECT**, **FROM**, and **WHERE** clauses match more than 10,000 metrics, the query only processes the first 10,000 of these metrics that it finds.
- A single query can return no more than 500 time series. This means that if the query would return more than 500 metrics, not all metrics will be returned in the query results. If you use an **ORDER BY** clause, then all the metrics being processed are sorted, and the 500 that have the highest or lowest values according to your **ORDER BY** clause are returned.

  If you do not include an **ORDER BY** clause, you can't control which 500 matching metrics are returned.
- You can have as many as 200 Metrics Insights alarms per Region.
- Metrics Insights does not support high-resolution data, which is metric data reported with a granularity of less than one minute. If you request high-resolution data, the request does not fail, but the output is aggregated at one-minute granularity.
- Each GetMetricData operation can have only one query, but you can have multiple widgets in a dashboard that each include a query.
- If a query using tag(s) with a **GROUP BY** or **WHERE** matches a metric that has more than 10 tag updates, only the most recent 10 tagged versions of the metric will be included in the query results.

Metrics Insights sample queries

This section contains examples of useful CloudWatch Metrics Insights queries that you can copy and use directly or copy and modify in query editor. Some of these examples are already available in the console, and you can access them by choosing **Add query** in the **Metrics** view.

Application Load Balancer examples

Total requests across all load balancers

```
SELECT SUM(RequestCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
```

Top 10 most active load balancers

```
SELECT MAX(ActiveConnectionCount)
FROM SCHEMA("AWS/ApplicationELB", LoadBalancer)
GROUP BY LoadBalancer
ORDER BY SUM() DESC
LIMIT 10
```

AWS API usage examples

Top 20 AWS APIs by the number of calls in your account

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API'
GROUP BY Service, Resource
ORDER BY COUNT() DESC
LIMIT 20
```

CloudWatch APIs sorted by calls

```
SELECT COUNT(CallCount)
FROM SCHEMA("AWS/Usage", Class, Resource, Service, Type)
WHERE Type = 'API' AND Service = 'CloudWatch'
GROUP BY Resource
ORDER BY COUNT() DESC
```

DynamoDB examples

Top 10 tables by consumed reads

```
SELECT SUM(ProvisionedWriteCapacityUnits)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by returned bytes

```
SELECT SUM(ReturnedBytes)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 tables by user errors

```
SELECT SUM(UserErrors)
FROM SCHEMA("AWS/DynamoDB", TableName)
GROUP BY TableName
ORDER BY MAX() DESC LIMIT 10
```

Amazon Elastic Block Store examples

Top 10 Amazon EBS volumes by bytes written

```
SELECT SUM(VolumeWriteBytes)
FROM SCHEMA("AWS/EBS", VolumeId)
GROUP BY VolumeId
ORDER BY SUM() DESC
LIMIT 10
```

Average Amazon EBS volume write time

```
SELECT AVG(VolumeTotalWriteTime)
FROM SCHEMA("AWS/EBS", VolumeId)
```

Amazon EC2 examples

CPU utilization of EC2 instances sorted by highest

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY AVG() DESC
```

Average CPU utilization across the entire fleet

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
```

Top 10 instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/EC2", InstanceId)
GROUP BY InstanceId
ORDER BY MAX() DESC
LIMIT 10
```

In this case, the CloudWatch agent is collecting a `CPUUtilization` metric per application. This query filters the average of this metric for a specific application name.

```
SELECT AVG(CPUUtilization)
FROM "AWS/CWAgent"
WHERE ApplicationName = 'eCommerce'
```

Amazon Elastic Container Service examples

Average CPU utilization across all ECS clusters

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
```

Top 10 clusters by memory utilization

```
SELECT AVG(MemoryUtilization)
FROM SCHEMA("AWS/ECS", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by CPU utilization

```
SELECT AVG(CPUUtilization)
FROM SCHEMA("AWS/ECS", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 services by running tasks (Container Insights)

```
SELECT AVG(RunningTaskCount)
FROM SCHEMA("ECS/ContainerInsights", ClusterName, ServiceName)
GROUP BY ClusterName, ServiceName
ORDER BY AVG() DESC
LIMIT 10
```

Amazon Elastic Kubernetes Service Container Insights examples

Average CPU utilization across all EKS clusters

```
SELECT AVG(pod_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
```

Top 10 clusters by node CPU utilization

```
SELECT AVG(node_cpu_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 clusters by pod memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName)
GROUP BY ClusterName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 nodes by CPU utilization

```
SELECT AVG(node_cpu_utilization)
```

```
FROM SCHEMA("ContainerInsights", ClusterName, NodeName)
GROUP BY ClusterName, NodeName
ORDER BY AVG() DESC LIMIT 10
```

Top 10 pods by memory utilization

```
SELECT AVG(pod_memory_utilization)
FROM SCHEMA("ContainerInsights", ClusterName, PodName)
GROUP BY ClusterName, PodName
ORDER BY AVG() DESC LIMIT 10
```

EventBridge examples

Top 10 rules by invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by failed invocations

```
SELECT SUM(FailedInvocations)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Top 10 rules by matched rules

```
SELECT SUM(MatchedEvents)
FROM SCHEMA("AWS/Events", RuleName)
GROUP BY RuleName
ORDER BY MAX() DESC LIMIT 10
```

Kinesis examples

Top 10 streams by bytes written

```
SELECT SUM("PutRecords.Bytes")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
```

```
ORDER BY SUM() DESC LIMIT 10
```

Top 10 streams by earliest items in the stream

```
SELECT MAX("GetRecords.IteratorAgeMilliseconds")
FROM SCHEMA("AWS/Kinesis", StreamName)
GROUP BY StreamName
ORDER BY MAX() DESC LIMIT 10
```

Lambda examples

Lambda functions ordered by number of invocations

```
SELECT SUM(Invocations)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
```

Top 10 Lambda functions by longest runtime

```
SELECT AVG(Duration)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Lambda functions by error count

```
SELECT SUM(Errors)
FROM SCHEMA("AWS/Lambda", FunctionName)
GROUP BY FunctionName
ORDER BY SUM() DESC
LIMIT 10
```

CloudWatch Logs examples

Top 10 log groups by incoming events

```
SELECT SUM(IncomingLogEvents)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
```

```
ORDER BY SUM() DESC LIMIT 10
```

Top 10 log groups by written bytes

```
SELECT SUM(IncomingBytes)
FROM SCHEMA("AWS/Logs", LogGroupName)
GROUP BY LogGroupName
ORDER BY SUM() DESC LIMIT 10
```

Amazon RDS examples

Top 10 Amazon RDS instances by highest CPU utilization

```
SELECT MAX(CPUUtilization)
FROM SCHEMA("AWS/RDS", DBInstanceIdentifier)
GROUP BY DBInstanceIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Top 10 Amazon RDS clusters by writes

```
SELECT SUM(WriteIOPS)
FROM SCHEMA("AWS/RDS", DBClusterIdentifier)
GROUP BY DBClusterIdentifier
ORDER BY MAX() DESC
LIMIT 10
```

Amazon Simple Storage Service examples

Average latency by bucket

```
SELECT AVG(TotalRequestLatency)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY AVG() DESC
```

Top 10 buckets by bytes downloaded

```
SELECT SUM(BytesDownloaded)
FROM SCHEMA("AWS/S3", BucketName, FilterId)
```

```
WHERE FilterId = 'EntireBucket'
GROUP BY BucketName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon Simple Notification Service examples

Total messages published by SNS topics

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
```

Top 10 topics by messages published

```
SELECT SUM(NumberOfMessagesPublished)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 topics by message delivery failures

```
SELECT SUM(NumberOfNotificationsFailed)
FROM SCHEMA("AWS/SNS", TopicName)
GROUP BY TopicName
ORDER BY SUM() DESC
LIMIT 10
```

Amazon SQS examples

Top 10 queues by number of visible messages

```
SELECT AVG(ApproximateNumberOfMessagesVisible)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Top 10 most active queues

```
SELECT SUM(NumberOfMessagesSent)
```

```
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY SUM() DESC
LIMIT 10
```

Top 10 queues by age of earliest message

```
SELECT AVG(ApproximateAgeOfOldestMessage)
FROM SCHEMA("AWS/SQS", QueueName)
GROUP BY QueueName
ORDER BY AVG() DESC
LIMIT 10
```

Metrics Insights glossary

label

In Metrics Insights, a label is a key-value pair that is used to scope a query to return a particular set of data, or to define criteria by which query results are to be separated into separate time series. A label key is similar to a column name in SQL. Labels must be CloudWatch metric dimensions.

observation

An observation is a value recorded for a given metric at a given time.

Troubleshooting Metrics Insights

The results include "Other," but I don't have this as a dimension

This means that the query includes a **GROUP BY** clause that specifies a label key that is not used in some of the metrics that are returned by the query. In this case, a null group named `Other` is returned. The metrics that do not include that label key are probably aggregated metrics that return values aggregated across all values of that label key.

For example, suppose you have the following query:

```
SELECT AVG(Faults)
FROM MyCustomNamespace
GROUP BY Operation, ServiceName
```

If some of the returned metrics don't include `ServiceName` as a dimension, then those metrics are displayed as having `Other` as the value for `ServiceName`.

To prevent seeing "Other" in your results, use **SCHEMA** in your **FROM** clause, as in the following example:

```
SELECT AVG(Faults)
FROM SCHEMA(MyCustomNamespace, Operation)
GROUP BY Operation, ServiceName
```

This limits the returned results to only the metrics that have both the `Operation` and `ServiceName` dimensions.

The oldest timestamp in my graph has a lower metric value than the others

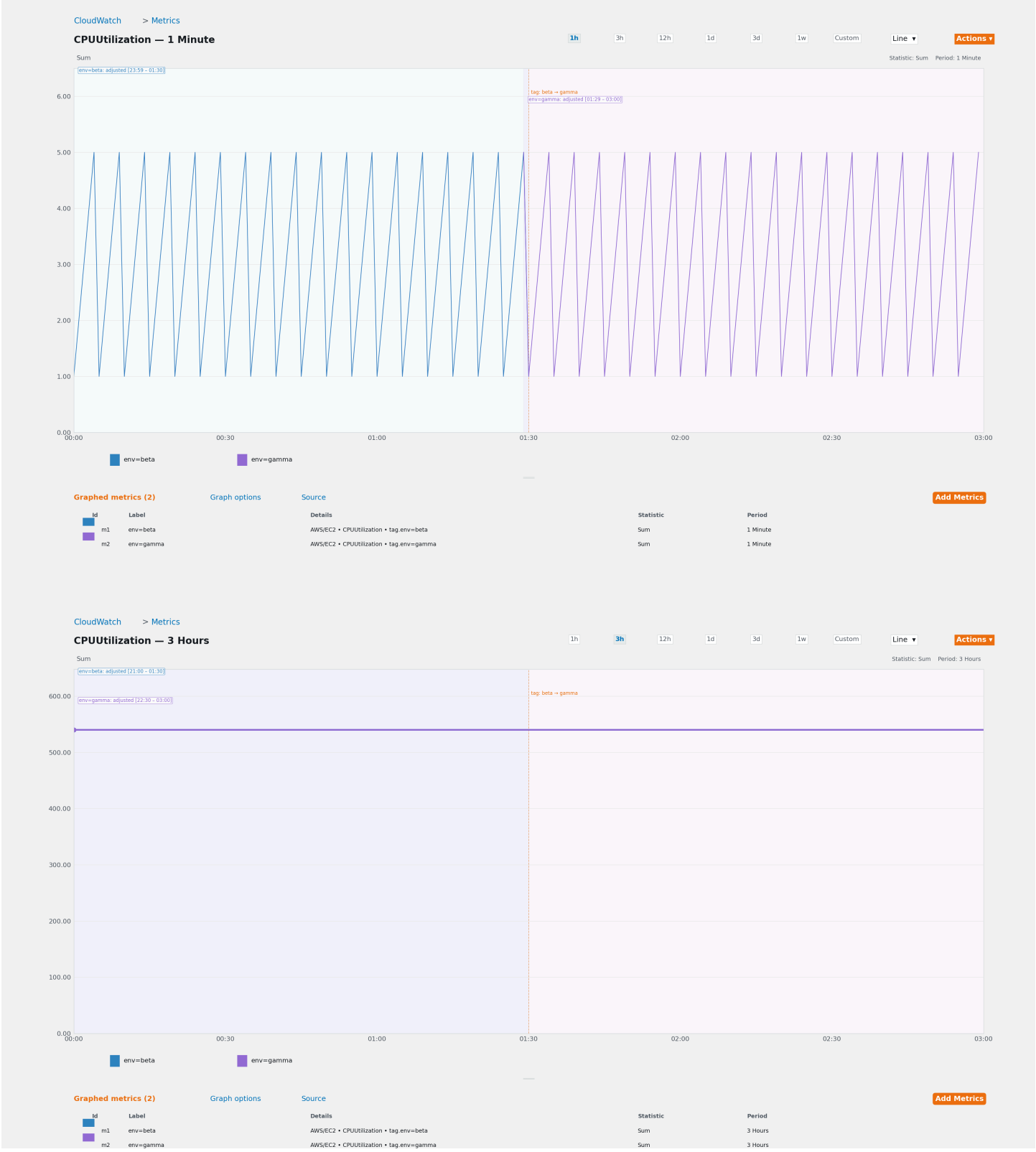
CloudWatch Metrics Insights supports up to two weeks of historical data. When you graph with a period larger than one minute, there could be cases where the oldest data point differs from the expected value. This is because the CloudWatch Metrics Insights queries return only data within the two-week retention period. In this case, the oldest data point in the query returns only the observations that have been measured within the two-week boundary, instead of returning all the observations within the period of that data point.

Inconsistent metric values across different time periods when using tag-based queries

When you use `WHERE` or `GROUP BY` clauses with tags in CloudWatch Metrics Insights queries, you might see different metric values depending on the selected time period. For example, a 6-hour period might show a peak value of 20, while a 1-hour period shows only 2 for the same time window.

This occurs because tag timestamps are stored with second-level resolution, while metric data points are aligned to period boundaries (for example, the start of each minute or hour). To determine which data points match a tag time range, CloudWatch adjusts the start of the range by subtracting one period. With larger periods, this adjustment creates a wider gap between the tag timestamp and the earliest included data point, which can cause data points near the start of the range to be excluded.

The following example shows how this affects query results. A metric has two tag values: `env=beta` (from 00:00 to 01:30) and `env=gamma` (from 01:30 to 03:00). Each tag covers 90 minutes of data with a SUM of 270.


CloudWatch > Metrics
CPUUtilization — 1 Minute
1h 3h 12h 1d 3d 1w Custom Line Actions
Sum
Statistic: Sum Period: 1 Minute
env=beta: adjusted [23:59 – 01:30]
tag: beta → gamma
env=gamma: adjusted [01:29 – 03:00]
6.00
5.00
4.00
3.00
2.00
1.00
0.00
00:00 00:30 01:00 01:30 02:00 02:30 03:00
env=beta
env=gamma
Graphed metrics (2)
Graph options
Source
Add Metrics
Id Label Details Statistic Period
m1 env=beta AWS/EC2 • CPUUtilization • tag.env=beta Sum 1 Minute
m2 env=gamma AWS/EC2 • CPUUtilization • tag.env=gamma Sum 1 Minute
CloudWatch > Metrics
CPUUtilization — 3 Hours
1h 3h 12h 1d 3d 1w Custom Line Actions
Sum
Statistic: Sum Period: 3 Hours
env=beta: adjusted [21:00 – 01:30]
env=gamma: adjusted [22:30 – 03:00]
tag: beta → gamma
600.00
500.00
400.00
300.00
200.00
100.00
0.00
00:00 00:30 01:00 01:30 02:00 02:30 03:00
env=beta
env=gamma
Graphed metrics (2)
Graph options
Source
Add Metrics
Id Label Details Statistic Period
m1 env=beta AWS/EC2 • CPUUtilization • tag.env=beta Sum 3 Hours
m2 env=gamma AWS/EC2 • CPUUtilization • tag.env=gamma Sum 3 Hours

env=beta with 1-minute period			
Statistic	**Expected**	**Returned**	**Difference**
SUM	270	271	+1
AVG	3.0	3.0	0
MIN	1	1	0
MAX	5	5	0
SAMPLE_COUNT	90	91	+1

env=gamma with 1-minute period			
Statistic	**Expected**	**Returned**	**Difference**
SUM	270	275	+5
AVG	3.0	3.0	0
MIN	1	1	0
MAX	5	5	0
SAMPLE_COUNT	90	91	+1

With a 1-minute period, the alignment adjustment is small (1 minute), so only 1 extra data point is included per tag. With a 3-hour period, the adjustment spans the entire query range:

env=beta with 3-hour period			
Statistic	**Expected**	**Returned**	**Difference**
SUM	270	540	+270
AVG	3.0	3.0	0

env=beta with 3-hour period			
Statistic	**Expected**	**Returned**	**Difference**
MIN	1	1	0
MAX	5	5	0
SAMPLE_COUNT	90	180	+90

env=gamma with 3-hour period			
Statistic	**Expected**	**Returned**	**Difference**
SUM	270	540	+270
AVG	3.0	3.0	0
MIN	1	1	0
MAX	5	5	0
SAMPLE_COUNT	90	180	+90

With the 3-hour period, both tags return the entire dataset (SUM=540, SAMPLE_COUNT=180) because the single aggregated data point's timestamp falls within both adjusted ranges. The tag boundary is effectively erased.

To reduce the impact of this behaviour, try the following approaches:

- **Use smaller aggregation periods.** Smaller periods (such as 1 minute or 5 minutes) more closely match the second-level resolution of tag timestamps, which minimizes the alignment gap and makes it more likely that all relevant data points are included.
- **Use dimension-based filtering instead of tags.** If your use case allows it, filter by dimensions rather than tags. Dimension-based queries are not affected by this behaviour. For example, use `WHERE InstanceId = 'i-1234567890abcdef0'` instead of `WHERE tag."my-tag" = 'my-value'`.

- **Query at a consistent granularity.** When comparing metric data across different time windows, use the same period to avoid unexpected differences caused by the alignment adjustment.

View available metrics

Metrics are grouped first by namespace, and then by the various dimension combinations within each namespace. For example, you can view all EC2 metrics, EC2 metrics grouped by instance, or EC2 metrics grouped by Auto Scaling group.

Only the AWS services that you're using send metrics to Amazon CloudWatch.

For a list of AWS services that send metrics to CloudWatch, see AWS services that publish CloudWatch metrics. From this page, you can also see the metrics and dimensions that are published by each of those services.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in the search box in the **Browse** tab in the console. They are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the AWS CLI.
> If the old metric you want to view has a current metric with similar dimensions, you can view that current similar metric and then choose the **Source** tab, and change the metric name and dimension fields to the ones that you want, and also change the time range to a time when the metric was being reported.

The following steps help you browse through the metric namespaces to find and view metrics. You can also search for metrics using targeted search terms. For more information, see Search for available metrics.

If you are browsing in an account set up as a monitoring account in CloudWatch cross-account observability, you can view metrics from the source accounts linked to this monitoring account. When metrics from source accounts are displayed, the ID or label of the account that they are from is also displayed. For more information, see CloudWatch cross-account observability.

> **Note**
>
> After the source account stops sharing the metrics with the *Monitoring* account, the *Source* account metrics data is not accessible to the monitoring account. Source metric names can be visible to the monitoring account for upto 14 days.

To view available metrics by namespace and dimension using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Select a metric namespace (for example, **EC2** or **Lambda**).
4. Select a metric dimension (for example, **Per-Instance Metrics** or **By Function Name**).
5. The **Browse** tab displays all metrics for that dimension in the namespace. By each metric name is an information button you can choose to see a pop-up with the metric definition.

   If this is a monitoring account in CloudWatch cross-account observability, you also see the metrics from the source accounts linked to this monitoring account. The **Account label** and **Account id** columns in the table display which account each metric is from.

   You can do the following:

   a. To sort the table, use the column heading.
   b. To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
   c. To filter by account, choose the account label or account ID and then choose **Add to search**.
   d. To filter by resource, choose the resource ID and then choose **Add to search**.
   e. To filter by metric, choose the metric name and then choose **Add to search**.
6. (Optional) To add this graph to a CloudWatch dashboard, choose **Actions**, **Add to dashboard**.

To view available metrics by account namespace, dimension, or metric using the AWS CLI

Use the list-metrics command to list CloudWatch metrics. For a list of the namespaces, metrics, and dimensions for all services that publish metrics, see AWS services that publish CloudWatch metrics.

The following example command lists all the metrics for Amazon EC2.

```
aws cloudwatch list-metrics --namespace AWS/EC2
```

The following is example output.

```
{
"Metrics" : [
  ...
  {
      "Namespace": "AWS/EC2",
      "Dimensions": [
          {
              "Name": "InstanceId",
              "Value": "i-1234567890abcdef0"
          }
      ],
      "MetricName": "NetworkOut"
  },
  {
      "Namespace": "AWS/EC2",
      "Dimensions": [
          {
              "Name": "InstanceId",
              "Value": "i-1234567890abcdef0"
          }
      ],
      "MetricName": "CPUUtilization"
  },
  {
      "Namespace": "AWS/EC2",
      "Dimensions": [
          {
              "Name": "InstanceId",
              "Value": "i-1234567890abcdef0"
          }
      ],
      "MetricName": "NetworkIn"
  },
  ...
]
}
```

To list all the available metrics for a specified resource

The following example specifies the AWS/EC2 namespace and the InstanceId dimension to view the results for the specified instance only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --dimensions
 Name=InstanceId,Value=i-1234567890abcdef0
```

To list a metric for all resources

The following example specifies the AWS/EC2 namespace and a metric name to view the results for the specified metric only.

```
aws cloudwatch list-metrics --namespace AWS/EC2 --metric-name CPUUtilization
```

To retrieve metrics from linked source accounts in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from both the monitoring account and all linked source accounts. If you do not add --include-linked-accounts, the command returns only the monitoring account's metrics.

```
aws cloudwatch list-metrics --include-linked-accounts
```

To retrieve metrics from a source account in CloudWatch cross-account observability

The following example is run in a monitoring account to retrieve metrics from the source account with the ID 111122223333.

```
aws cloudwatch list-metrics --include-linked-accounts --owning-account "111122223333"
```

Search for available metrics

You can search within all of the metrics in your account using targeted search terms. Metrics are returned that have matching results within their namespace, metric name, or dimensions.

If this is a monitoring account in CloudWatch cross-account observability, you also search for metrics from the source accounts linked to this monitoring account.

> **Note**
>
> Metrics that have not had any new data points in the past two weeks do not appear in the console. They also do not appear when you type their metric name or dimension names in

the search box in the **Browse** tab in the console. They are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the get-metric-data or get-metric-statistics commands in the AWS CLI.

To search for available metrics in CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the search field, enter a search term, such as a metric name, namespace, account ID, account label, dimension name or value, or resource name. This shows you all of the namespaces with metrics with this search term.

   For example, if you search for **`volume`**, this shows the namespaces that contain metrics with this term in their name.

   For more information on search, see Use search expressions in graphs
4. To graph all the search results, choose **Graph search**

   or

   Select a namespace to view the metrics from that namespace. You can then do the following:

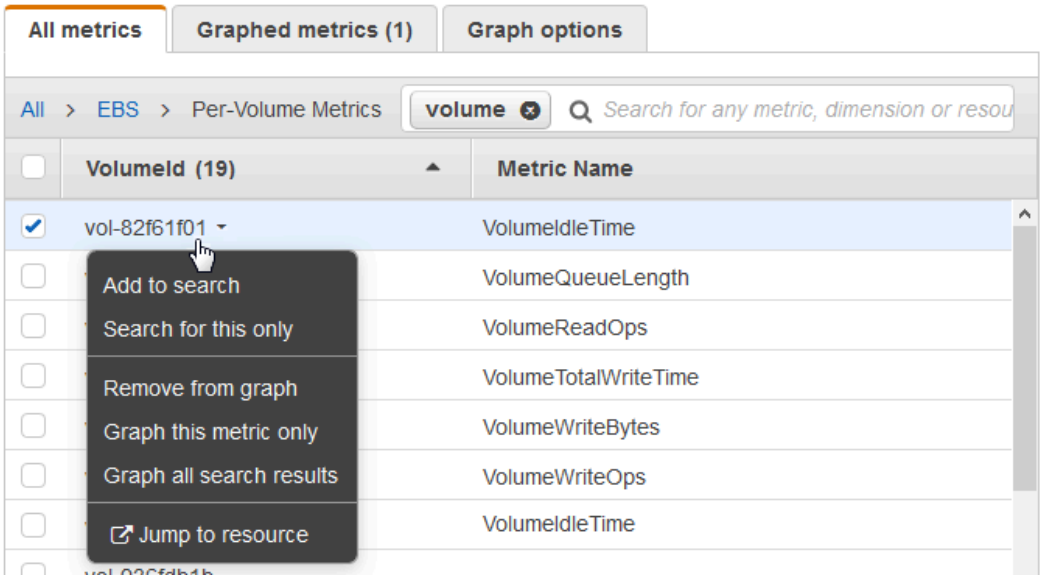
   a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.

   b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.

   c. To view one of the resources on its console, choose the resource ID and then choose **Jump to resource**.

   d. To view help for a metric, select the metric name and choose **What is this?**.

   The selected metrics appear on the graph.



5. (Optional) Select one of the buttons in the search bar to edit that part of the search term.

Retrieve metric data (GetMetricData)

`GetMetricData` is the primary API for retrieving Classic metric data points programmatically. It supports batch queries, math expressions, and queries in a single call.

Basic usage

Retrieve a single metric:

```
aws cloudwatch get-metric-data \
  --metric-data-queries '[
    {
      "Id": "cpu",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/EC2",
          "MetricName": "CPUUtilization",
          "Dimensions": [{"Name": "InstanceId", "Value": "i-1234567890abcdef0"}]
        },
        "Period": 300,
        "Stat": "Average"
      }
    }
  ]' \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
```

```
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --output json
```

Batch queries

Retrieve multiple metrics in a single call (up to 500 queries):

```
aws cloudwatch get-metric-data \
  --metric-data-queries '[
    {"Id": "cpu", "MetricStat": {"Metric": {"Namespace": "AWS/EC2", "MetricName":
 "CPUUtilization", "Dimensions": [{"Name": "InstanceId", "Value": "i-abc"}]}, "Period":
 300, "Stat": "Average"}},
    {"Id": "mem", "MetricStat": {"Metric": {"Namespace": "CWAgent", "MetricName":
 "mem_used_percent", "Dimensions": [{"Name": "InstanceId", "Value": "i-abc"}]},
 "Period": 300, "Stat": "Average"}}
  ]' \
  --start-time 2024-01-01T00:00:00 \
  --end-time 2024-01-01T01:00:00
```

Math expressions

Combine metrics using arithmetic:

```
[
  {"Id": "rx", "MetricStat": {"Metric": {"Namespace": "AWS/EC2", "MetricName":
 "NetworkIn", "Dimensions": [{"Name": "InstanceId", "Value": "i-abc"}]}, "Period": 300,
 "Stat": "Sum"}},
  {"Id": "tx", "MetricStat": {"Metric": {"Namespace": "AWS/EC2", "MetricName":
 "NetworkOut", "Dimensions": [{"Name": "InstanceId", "Value": "i-abc"}]}, "Period":
 300, "Stat": "Sum"}},
  {"Id": "total", "Expression": "rx + tx", "Label": "TotalNetwork"}
]
```

GetMetricData vs GetMetricStatistics

	GetMetricData	GetMetricStatistics
Batch support	Up to 500 queries per call	Single metric per call
Math expressions	Yes	No

	GetMetricData	GetMetricStatistics
queries	Yes	No
Pagination	Yes (NextToken)	No
Recommended	Yes — primary API	Legacy — use GetMetricData instead

Pricing

$0.01 per 1,000 metrics requested through `GetMetricData`.

> **Note**
>
> For OTel metrics, use the Prometheus-compatible query API or CloudWatch Query Studio with PromQL. `GetMetricData` does not query OTel metrics.

Get statistics for a metric (GetMetricStatistics)

`GetMetricStatistics` is the legacy single-metric API. For new code, use Retrieve metric data (GetMetricData) instead — it supports batch queries and math expressions.

Usage

```
aws cloudwatch get-metric-statistics \
  --namespace "AWS/EC2" \
  --metric-name "CPUUtilization" \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Average Maximum
```

CloudWatch statistics definitions

Statistics are metric data aggregations over specified periods of time. When you graph or retrieve the statistics for a metric, you specify the *Period* of time, such as five minutes, to use to calculate

each statistical value. For example, if the **Period** is five minutes, the **Sum** is the sum of all sample values collected during the five-minute period, while the **Minimum** is the lowest value collected during the five-minute period.

CloudWatch supports the following statistics for metrics.

- **SampleCount** is the number of data points during the period.
- **Sum** is the sum of the values of all the data points collected during the period.
- **Average** is the value of `Sum/SampleCount` during the specified period.
- **Minimum** is the lowest value observed during the specified period.
- **Maximum** is the highest value observed during the specified period.
- **Percentile (p)** indicates the relative standing of a value in a dataset. For example, **p95** is the 95th percentile and means that 95 percent of the data within the period is lower than this value and 5 percent of the data is higher than this value. Percentiles help you get a better understanding of the distribution of your metric data.
- **Trimmed mean (TM)** is the mean of all values that are between two specified boundaries. Values outside of the boundaries are ignored when the mean is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. The numbers can be absolute values or percentages. For example, **tm90** calculates the average after removing the 10% of data points with the highest values. **TM(2%:98%)** calculates the average after removing the 2% lowest data points and the 2% highest data points. **TM(150:1000)** calculates the average after removing all data points that are lower than or equal to 150, or higher than 1000.
- **Interquartile mean (IQM)** is the trimmed mean of the *interquartile range*, or the middle 50% of values. It is equivalent to **TM(25%:75%)**.
- **Winsorized mean (WM)** is similar to trimmed mean. However, with winsorized mean, the values that are outside the boundary are not ignored, but instead are considered to be equal to the value at the edge of the appropriate boundary. After this normalization, the average is calculated. You define the boundaries as one or two numbers between 0 and 100, up to 10 decimal places. For example, **wm98** calculates the average while treating the 2% of the highest values to be equal to the value at the 98th percentile. **WM(10%:90%)** calculates the average while treating the highest 10% of data points to be the value of the 90% boundary, and treating the lowest 10% of data points to be the value of the 10% boundary.
- **Percentile rank (PR)** is the percentage of values that meet a fixed threshold. For example, **PR(:300)** returns the percentage of data points that have a value of 300 or less. **PR(100:2000)** returns the percentage of data points that have a value between 100 and 2000.

Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

- **Trimmed count (TC)** is the number of data points in the chosen range for a trimmed mean statistic. For example, **tc90** returns the number of data points not including any data points that fall in the highest 10% of the values. **TC(0.005:0.030)** returns the number of data points with values between 0.005 (exclusive) and 0.030 (inclusive).

  Trimmed count can return a decimal value instead of an integer. This is because it is an interpolated, approximate value and can give fractional results.
- **Trimmed sum (TS)** is the sum of the values of data points in a chosen range for a trimmed mean statistic. It is equivalent to (Trimmed Mean) * (Trimmed count). For example, **ts90** returns the sum of the data points not including any data points that fall in the highest 10% of the values. **TS(80%:)** returns the sum of the data point values, not including any data points with values in the lowest 80% of the range of values.

> **Note**
>
> For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, if you define two boundaries as fixed values instead of percentages, the calculation includes values equal to the higher boundary, but does not include values equal to the lower boundary.

Syntax

For Trimmed Mean, Trimmed Count, Trimmed Sum, and Winsorized Mean, the following syntax rules apply:

- Using parentheses with one or two numbers with percent signs defines the boundaries to use as the values in the data set that fall in between the two percentiles that you specify. For example, **TM(10%:90%)** uses only the values between the 10th and 90th percentiles. **TM(:95%)** uses the values from the lowest end of the data set up to the 95th percentile, ignoring the 5% of data points with the highest values.
- Using parenthesis with one or two numbers without percent signs defines the boundaries to use as the values in the data set that fall in between the explicit values that you specify. For example, **TC(80:500)** uses only the values that are between 80 (exclusive) and 500 (inclusive). **TC(:0.5)** uses only the values that equal 0.5 or are lower.

- Using one number without parentheses calculates using percentages, ignoring data points that are higher than the specified percentile. For example, **tm99** calculates the mean while ignoring the 1% of the data points with the highest value. It is the same as **TM(:99%)**.
- Trimmed mean, Trimmed Count, Trimmed Sum, and Winsorized Mean can all be abbreviated using uppercase letters when specifying a range, such as **TM(5%:95%)**, **TM(100:200)**, or **TM(:95%)**. They can only be abbreviated using lowercase letters when you specifying only one number, such as **tm99**.

Statistics use cases

- **Trimmed mean** is most useful for metrics with a large sample size, such as webpage latency. For example, **tm99** disregards extreme high outliers that could result from network problems or human errors, to give a more accurate number for the average latency of typical requests. Similarly, **TM(10%:)** disregards the lowest 10% of latency values, such as those resulting from cache hits. And **TM(10%:99%)** excludes both of these types of outliers. We recommend that you use trimmed mean for monitoring latency.
- It is a good idea to keep watch on trimmed count whenever you are using trimmed mean, to make sure that the number of values being used in your trimmed mean calculations are enough to be statistically significant.
- Percentile rank enables you to put values into "bins" of ranges, and you can use this to manually create a histogram. To do this, break your values down into various bins, such as **PR(:1)**, **PR(1:5)**, **PR(5:10)**, and **PR(10:)**. Put each of these bins into a visualization as bar charts, and you have a histogram.

  Percentile rank is exclusive on the lower bound and inclusive on the upper bound.

Percentiles versus trimmed mean

A percentile such as **p99** and a trimmed mean such as **tm99** measure similar, but not identical values. Both **p99** and **tm99** ignore the 1% of the data points with the highest values, which are considered outliers. After that, **p99** is the maximum value of the remaining 99%, while **tm99** is the *average* of the remaining 99%. If you are looking at the latency of web requests, **p99** tells you the worst customer experience, ignoring outliers, while **tm99** tells you the average customer experience, ignoring outliers.

Trimmed mean is a good latency statistic to watch if you are looking to optimize your customer experience.

Requirements to use percentiles, trimmed mean, and some other statistics

CloudWatch needs raw data points to calculate the following statistics:

- Percentiles
- Trimmed mean
- Interquartile mean
- Winsorized mean
- Trimmed sum
- Trimmed count
- Percentile rank

If you publish data for a custom statistics using a statistic set instead of raw data, you can retrieve these types of statistics for this data only if one of the following conditions is true:

- The SampleCount value of the statistic set is 1 and Min, Max, and Sum are all equal.
- The Min and Max are equal, and Sum is equal to Min multiplied by SampleCount.

The following AWS services include metrics that support these types of statistics.

- API Gateway
- Application Load Balancer
- Amazon EC2
- Elastic Load Balancing
- Kinesis
- Amazon RDS

Additionally, these type of statistics are not available for metrics when any of the metric values are negative numbers.

When to use this API

- Legacy applications that already depend on it
- Simple single-metric retrievals where batch isn't needed

For everything else, use Retrieve metric data (GetMetricData).

Use metrics explorer to monitor resources by their tags and properties

Metrics explorer is a tag-based tool that enables you to filter, aggregate, and visualize your metrics by tags and resource properties, to enhance observability for your services. This gives you a flexible and dynamic troubleshooting experience, so that you can create multiple graphs at a time and use these graphs to build your application health dashboards.

Metrics explorer visualizations are dynamic, so if a matching resource is created after you create a metrics explorer widget and add it to a CloudWatch dashboard, the new resource automatically appears in the explorer widget.

For example, if all of your EC2 production instances have the **`production`** tag, you can use metrics explorer to filter and aggregate metrics from all of these instances to understand their health and performance. If a new instance with a matching tag is later created, it's automatically added to the metrics explorer widget.

> **Note**
>
> Metrics explorer provides a point-in-time experience. Resources that have been terminated, or no longer exist with the property or tag that you specified are not displayed in the visualisation. However, you can still find the metrics for these resources in CloudWatch metrics views.

With metrics explorer, you can choose how to aggregate metrics from the resources that match the criteria, and whether to show them all in a single graph or on different graphs within one metrics explorer widget.

Metrics explorer includes templates that you can use to see useful visualization graphs with a single selection, and you can also extend these templates to create completely customized metrics explorer widgets.

You can use metrics explorer across accounts if you are using the Cross-account cross-Region CloudWatch console. However, metrics explorer is not supported for cross-account use from a monitoring account in CloudWatch cross-account observability.

Metrics explorer supports metrics emitted by AWS and EC2 metrics that are published by the CloudWatch agent, including memory, disk, and CPU metrics. To use metrics explorer to see the

metrics that are published by the CloudWatch agent, you might have to update your CloudWatch agent configuration file. For more information, see CloudWatch agent configuration for metrics explorer

To create a visualization with metrics explorer and optionally add it to a dashboard, follow these steps.

To create a visualization with metrics explorer

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Explorer**.
3. Do one of the following:
   - To use a template, select it in the box that currently shows **Empty Explorer**.

     Depending on the template, the explorer might immediately display graphs of metrics. If it doesn't, choose one or more tags or properties in the **From** box and then data should appear. If it doesn't, use the options at the top of the page to display a longer time range in the graphs.
   - To create a custom visualization, under **Metrics**, choose a single metric or all the available metrics from a service.

     After you choose a metric, you can optionally repeat this step to add more metrics.
4. For each metric selected, CloudWatch displays the statistic that it will use immediately after the metric name. To change this, choose the statistic name, and then choose the statistic that you want.
5. Under **From**, choose a tag or a resource property to filter your results.

   After you do this, you can optionally repeat this step to choose more tags or resource properties.

   If you choose multiple values of the same property, such as two EC2 instance types, the explorer displays all the resources that match either chosen property. It's treated as an OR operation.

   If you choose different properties or tags, such as the **`Production`** tag and the M5 instance type, only the resources that match all of these selections are displayed. It's treated as an AND operation.

6. (Optional) For **Aggregate by**, choose a statistic to use to aggregate the metrics. Then, next to **for**, choose how to aggregate the metric from the list. You can aggregate together all the resources that are currently displayed, or aggregate by a single tag or resource property.

   Depending on how you choose to aggregate, the result may be a single time series or multiple time series.
7. Under **Split by**, you can choose to split a single graph with multiple time series into multiple graphs. The split can be made by a variety of criteria, which you choose under **Split by**.
8. Under **Graph options**, you can refine the graph by changing the period, the type of graph, the legend placement, and the layout.
9. To add this visualization as a widget to a CloudWatch dashboard, choose **Add to dashboard**.

CloudWatch agent configuration for metrics explorer

To enable metrics explorer to discover EC2 metrics published by the CloudWatch agent, make sure that the CloudWatch agent configuration file contains the following values:

- In the `metrics` section, make sure that the `aggregation_dimensions` parameter includes `["InstanceId"]`. It can also contain other dimensions.
- In the `metrics` section, make sure that the `append_dimensions` parameter includes a `{"InstanceId":"${aws:InstanceId}"}` line. It can also contain other lines.
- In the `metrics` section, inside the `metrics_collected` section, check the sections for each resource type that you want metrics explorer to discover, such as the `cpu`, `disk`, and `memory` sections. Make sure that each of these sections has a `"resources": [ "*"] line.`.
- In the `cpu` section of the `metrics_collected` section, make sure there is a `"totalcpu": true` line.
- You must use the default `CWAgent` namespace for the metrics collected by the CloudWatch agent, instead of a custom namespace.

The settings in the previous list cause the CloudWatch agent to publish aggregate metrics for disks, CPUs, and other resources that can be plotted in metrics explorer for all the instances that use it.

These settings will republish the metrics that you had previously set up to be published with multiple dimensions, adding to your metric costs.

For more information about editing the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file.

Use search expressions in graphs

Search expressions are a type of math expression that you can add to CloudWatch graphs. Search expressions enable you to quickly add multiple related metrics to a graph. They also enable you to create dynamic graphs that automatically add appropriate metrics to their display, even if those metrics don't exist when you first create the graph.

For example, you can create a search expression that displays the `AWS/EC2 CPUUtilization` metric for all instances in the Region. If you later launch a new instance, the `CPUUtilization` of the new instance is automatically added to the graph.

When you use a search expression in a graph, the search finds the search expression in metric names, namespaces, dimension names, and dimension values. You can use Boolean operators for more complex and powerful searches. A search expression can find only metrics that have reported data within the past two weeks.

You can't create an alarm based on the **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

If you are using a monitoring account in CloudWatch cross-account observability, your search expressions can find metrics in the source accounts linked to that monitoring account.

Topics

- CloudWatch search expression syntax
- CloudWatch search expression examples
- Create a CloudWatch graph with a search expression

CloudWatch search expression syntax

A valid search expression has the following format.

```
SEARCH(' {Namespace, DimensionName1, DimensionName2, ...} SearchTerm', 'Statistic')
```

For example:

```
SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')
```

- The first part of the query after the word SEARCH, enclosed in curly braces, is the *metric schema* to be searched. The metric schema contains a metric namespace and one or more dimension names. Including a metric schema in a search query is optional. If specified, the metric schema must contain a namespace and can optionally contain one or more dimension names that are valid in that namespace.

  You don't need to use quote marks inside the metric schema unless a namespace or dimension name includes spaces or non-alphanumeric characters. In that case, you must enclose the name that contains those characters with double quotes.

- The `SearchTerm` is also optional, but a valid search must contain either the metric schema, the `SearchTerm`, or both. The `SearchTerm` usually contains one or more account IDs, metric names or dimension values. The `SearchTerm` can include multiple terms to search for, by both partial match and exact match. It can also contain Boolean operators.

  Using an account ID in a `SearchTerm` works only in accounts that are set up as monitoring accounts for CloudWatch cross-account observability. The syntax for an account ID in `SearchTerm` is `:aws.AccountId = 444455556666`. You can also use `'LOCAL'` to specify the monitoring account itself: `:aws.AccountId = 'LOCAL'`

  For more information, see CloudWatch cross-account observability.

  The `SearchTerm` can include one or more designators, such as `MetricName=` as in this example, but using designators isn't required.

  The metric schema and `SearchTerm` must be enclosed together in a pair of single quote marks.

- The `Statistic` is the name of any valid CloudWatch statistic. It must be enclosed by single quotes. For more information, see Statistics.

The preceding example searches the `AWS/EC2` namespace for any metrics that have `InstanceId` as a dimension name. It returns all `CPUUtilization` metrics that it finds, with the graph showing the `Average` statistic.

A search expression can find only metrics that have reported data within the past two weeks.

Search expression limits

The maximum search expression query size is 1024 characters. You can have as many as 100 search expressions on one graph. A graph can display as many as 500 time series.

CloudWatch search expressions: Tokenization

When you specify a `SearchTerm`, the search function searches for *tokens*, which are substrings that CloudWatch automatically generates from full metric names, dimension names, dimension values, and namespaces. CloudWatch generates tokens distinguished by the camel-case capitalization in the original string. Numeric characters also serve as the start of new tokens, and non-alphanumeric characters serve as delimiters, creating tokens before and after the non-alphanumeric characters.

A continuous string of the same type of token delimiter character results in one token.

All generated tokens are in lowercase. The following table shows some examples of tokens generated.

Original string	Tokens generated
CustomCount1	`customcount1` , `custom`, `count`, `1`
SDBFailure	`sdbfailure` , `sdb`, `failure`
Project2-trial333	`project2trial333` , `project`, `2`, `trial`, `333`

CloudWatch search expressions: Partial matches

When you specify a `SearchTerm`, the search term is also tokenized. CloudWatch finds metrics based on partial matches, which are matches of a single token generated from the search term to a single token generated from a metric name, namespace, dimension name, or dimension value.

Partial match searches to match a single token are case insensitive. For example, using any of the following search terms can return the `CustomCount1` metric:

- `count`
- `Count`
- `COUNT`

However, using `couNT` as a search term doesn't find `CustomCount1` because the capitalization in the search term `couNT` is tokenized into `cou` and `NT`.

Searches can also match composite tokens, which are multiple tokens that appear consecutively in the original name. To match a composite token, the search is case sensitive. For example, if the original term is `CustomCount1`, searches for `CustomCount` or `Count1` are successful, but searches for `customcount` or `count1` aren't.

CloudWatch search expressions: Exact matches

You can define a search to find only exact matches of your search term by using double quotes around the part of the search term that requires an exact match. These double-quotes are enclosed in the single-quotes used around the entire search term. For example, **`SEARCH(' {MyNamespace}, "CustomCount1" ', 'Maximum')`** finds the exact string `CustomCount1` if it exists as a metric name, dimension name, or dimension value in the namespace named `MyNamespace`. However, the searches **`SEARCH(' {MyNamespace}, "customcount1" ', 'Maximum')`** or **`SEARCH(' {MyNamespace}, "Custom" ', 'Maximum')`** do not find this string.

You can combine partial match terms and exact match terms in a single search expression. For example, **`SEARCH(' {AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')`** returns the Elastic Load Balancing metric named `ConsumedLCUs` as well as all Elastic Load Balancing metrics or dimensions that contain the token `flow`.

Using exact match is also a good way to find names with special characters, such as non-alphanumeric characters or spaces, as in the following example.

```
SEARCH(' {"My Namespace", "Dimension@Name"}, "Custom:Name[Special_Characters" ',
 'Maximum')
```

CloudWatch search expressions: Excluding a metric schema

All examples shown so far include a metric schema, in curly braces. Searches that omit a metric schema are also valid.

For example, **`SEARCH(' "CPUUtilization" ', 'Average')`** returns all metric names, dimension names, dimension values, and namespaces that are an exact match for the string `CPUUtilization`. In the AWS metric namespaces, this can include metrics from several services including Amazon EC2, Amazon ECS, SageMaker AI, and others.

To narrow this search to only one AWS service, the best practice is to specify the namespace and any necessary dimensions in the metric schema, as in the following example. Although this narrows the search to the AWS/EC2 namespace, it would still return results of other metrics if you have defined CPUUtilization as a dimension value for those metrics.

```
SEARCH(' {AWS/EC2, InstanceType} "CPUUtilization" ', 'Average')
```

Alternatively you could add the namespace in the SearchTerm as in the following example. But in this example, the search would match any AWS/EC2 string, even if it was a custom dimension name or value.

```
SEARCH(' "AWS/EC2" MetricName="CPUUtilization" ', 'Average')
```

CloudWatch search expressions: Specifying property names in the search

The following exact match search for "CustomCount1" returns all metrics with exactly that name.

```
SEARCH(' "CustomCount1" ', 'Maximum')
```

But it also returns metrics with dimension names, dimension values, or namespaces of CustomCount1. To structure your search further, you can specify the property name of the type of object that you want to find in your searches. The following example searches all namespaces and returns metrics named CustomCount1.

```
SEARCH(' MetricName="CustomCount1" ', 'Maximum')
```

You can also use namespaces and dimension name/value pairs as property names, as in the following examples. The first of these examples also illustrates that you can use property names with partial match searches as well.

```
SEARCH(' InstanceType=micro ', 'Average')
```

```
SEARCH(' InstanceType="t2.micro" Namespace="AWS/EC2" ', 'Average')
```

CloudWatch search expressions: Non-alphanumeric characters

Non-alphanumeric characters serve as delimiters, and mark where the names of metrics, dimensions, namespaces, and search terms are to be separated into tokens. When terms are

tokenized, non-alphanumeric characters are stripped out and don't appear in the tokens. For example, `Network-Errors_2` generates the tokens `network`, `errors`, and `2`.

Your search term can include any non-alphanumeric characters. If these characters appear in your search term, they can specify composite tokens in a partial match. For example, all of the following searches would find metrics named either `Network-Errors-2` or `NetworkErrors2`.

```
network/errors
network+errors
network-errors
Network_Errors
```

When you're doing an exact value search, any non-alphanumeric characters used in the exact search must be the correct characters that appear in the string being searched for. For example, if you want to find `Network-Errors-2`, searching for `"Network-Errors-2"` is successful, but a search for `"Network_Errors_2"` isn't.

When you perform an exact match search, the following characters must be escaped with a backslash.

```
" \ ( )
```

For example, to find the metric name `Europe\France Traffic(Network)` by exact match, use the search term **`"Europe\\France Traffic\(Network\)"`**

CloudWatch search expressions: Boolean operators

Search supports the use of the Boolean operators `AND`, `OR`, and `NOT` within the `SearchTerm`. Boolean operators are enclosed in the single quote marks that you use to enclose the entire search term. Boolean operators are case sensitive, so `and`, `or`, and `not` aren't valid as Boolean operators.

You can use `AND` explicitly in your search, such as **`SEARCH('{AWS/EC2,InstanceId} network AND packets', 'Average')`**. Not using any Boolean operator between search terms implicitly searches them as if there were an `AND` operator, so **`SEARCH(' {AWS/EC2,InstanceId} network packets ', 'Average')`** yields the same search results.

Use `NOT` to exclude subsets of data from the results. For example, **`SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT i-1234567890123456 ', 'Average')`** returns the `CPUUtilization` for all your instances, except for the instance

i-1234567890123456. You can also use a NOT clause as the only search term. For example, **SEARCH('NOT Namespace=AWS ', 'Maximum')** yields all your custom metrics (metrics with namespaces that don't include AWS).

You can use multiple NOT phrases in a query. For example, **SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" NOT "ProjectA" NOT "ProjectB" ', 'Average')** returns the CPUUtilization of all instances in the Region, except for those with dimension values of ProjectA or ProjectB.

You can combine Boolean operators for more powerful and detailed searches, as in the following examples. Use parentheses to group the operators.

Both of the next two examples return all metric names containing ReadOps from both the EC2 and EBS namespaces.

```
SEARCH(' (EC2 OR EBS) AND MetricName=ReadOps ', 'Maximum')
```

```
SEARCH(' (EC2 OR EBS) MetricName=ReadOps ', 'Maximum')
```

The following example narrows the previous search to only results that include ProjectA, which could be the value of a dimension.

```
SEARCH(' (EC2 OR EBS) AND ReadOps AND ProjectA ', 'Maximum')
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions, and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH(' {AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

CloudWatch search expressions: Using math expressions

You can use a search expression within a math expressions in a graph.

For example, **SUM(SEARCH(' {AWS/Lambda, FunctionName} MetricName="Errors" ', 'Sum'))** returns the sum of the Errors metric of all your Lambda functions.

Using separate lines for your search expression and math expression might yield more useful results. For example, suppose that you use the following two expressions in a graph. The first line displays separate `Errors` lines for each of your Lambda functions. The ID of this expression is `e1`. The second line adds another line showing the sum of the errors from all of the functions.

```
SEARCH(' {AWS/Lambda, FunctionName}, MetricName="Errors" ', 'Sum')
SUM(e1)
```

CloudWatch search expression examples

The following examples illustrate more search expression uses and syntax. Let's start with a search for `CPUUtilization` across all instances in the Region and then look at variations.

This example displays one line for each instance in the Region, showing the `CPUUtilization` metric from the `AWS/EC2` namespace.

```
SEARCH(' {AWS/EC2,InstanceId} MetricName="CPUUtilization" ', 'Average')
```

Changing `InstanceId` to `InstanceType` changes the graph to show one line for each instance type used in the Region. Data from all instances of each type is aggregated into one line for that instance type.

```
SEARCH(' {AWS/EC2,InstanceType} MetricName="CPUUtilization" ', 'Average')
```

The following example aggregates the `CPUUtilization` by instance type and displays one line for each instance type that includes the string `micro`.

```
SEARCH('{AWS/EC2,InstanceType} InstanceType=micro MetricName="CPUUtilization" ',
 'Average')
```

This example narrows the previous example, changing the `InstanceType` to an exact search for t2.micro instances.

```
SEARCH('{AWS/EC2,InstanceType} InstanceType="t2.micro" MetricName="CPUUtilization" ',
 'Average')
```

The following search removes the `{metric schema}` part of the query, so the `CPUUtilization` metric from all namespaces appears in the graph. This can return quite a few results because the

graph includes multiple lines for the CPUUtilization metric from each AWS service, aggregated along different dimensions.

```
SEARCH('MetricName="CPUUtilization" ', 'Average')
```

To narrow these results a bit, you can specify two specific metric namespaces.

```
SEARCH('MetricName="CPUUtilization" AND ("AWS/ECS" OR "AWS/ES") ', 'Average')
```

The preceding example is the only way to do a search of specific multiple namespaces with one search query, as you can specify only one metric schema in each query. However, to add more structure, you could use two queries in the graph, as in the following example. This example also adds more structure by specifying a dimension to use to aggregate the data for Amazon ECS.

```
SEARCH('{AWS/ECS ClusterName}, MetricName="CPUUtilization" ', 'Average')
SEARCH(' {AWS/EBS} MetricName="CPUUtilization" ', 'Average')
```

The following example returns the Elastic Load Balancing metric named ConsumedLCUs as well as all Elastic Load Balancing metrics or dimensions that contain the token flow.

```
SEARCH('{AWS/NetworkELB, LoadBalancer} "ConsumedLCUs" OR flow ', 'Maximum')
```

The following example uses nested grouping. It returns Lambda metrics for Errors from all functions and Invocations of functions with names that include the strings ProjectA or ProjectB.

```
SEARCH('{AWS/Lambda,FunctionName} MetricName="Errors" OR (MetricName="Invocations" AND
 (ProjectA OR ProjectB)) ', 'Average')
```

The following example displays all of your custom metrics, excluding metrics generated by AWS services.

```
SEARCH('NOT Namespace=AWS ', 'Average')
```

The following example displays metrics with metric names, namespaces, dimension names, and dimension values that contain the string Errors as part of their name.

```
SEARCH('Errors', 'Average')
```

The following example narrows that search to exact matches. For example, this search finds the metric name `Errors` but not metrics named `ConnectionErrors` or `errors`.

```
SEARCH(' "Errors" ', 'Average')
```

The following example shows how to specify names that contain spaces or special characters in the metric schema part of the search term.

```
SEARCH('{"Custom-Namespace", "Dimension Name With Spaces"}, ErrorCount ', 'Maximum')
```

CloudWatch cross-account observability search expression examples

CloudWatch cross-account observability examples

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use the **SEARCH** function to return metrics from specified source accounts. For more information, see CloudWatch cross-account observability.

The following example retrieves all Lambda metrics from the account with the account ID 111122223333.

```
SEARCH(' AWS/Lambda :aws.AccountId = 111122223333 ', 'Average')
```

The following example retrieves all `AWS/EC2` metrics from two accounts: 111122223333 and 777788889999.

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 777788889999) ', 'Average')
```

The following example retrieves all `AWS/EC2` metrics from the source account 111122223333 and from the monitoring account itself.

```
SEARCH(' AWS/EC2 :aws.AccountId = (111122223333 OR 'LOCAL') ', 'Average')
```

The following example retrieves the `SUM` of the `MetaDataToken` metric from the account `444455556666` with the `InstanceId` dimension.

```
SEARCH('{AWS/EC2,InstanceId} :aws.AccountId=444455556666 MetricName=\"MetadataNoToken
\"','Sum')
```

Create a CloudWatch graph with a search expression

On the CloudWatch console, you can access search capability when you add a graph to a dashboard, or by using the **Metrics** view.

You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series.

To add a graph with a search expression to an existing dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards** and select a dashboard.
3. Choose **Add widget**.
4. Choose either **Line** or **Stacked area** and choose **Configure**.
5. On the **Graphed metrics** tab, choose **Add a math expression**.
6. For **Details**, enter the search expression that you want. For example, **`SEARCH('{AWS/EC2,InstanceId} MetricName="CPUUtilization"', 'Average')`**
7. (Optional) To add another search expression or math expression to the graph, choose **Add a math expression**
8. (Optional) After you add a search expression, you can specify a dynamic label to appear on the graph legend for each metric. Dynamic labels display a statistic about the metric and automatically update when the dashboard or graph is refreshed. To add a dynamic label, choose **Graphed metrics** and then **Dynamic labels**.

   By default, the dynamic values you add to the label appear at the beginning of the label. You can then choose the **Label** value for the metric to edit the label. For more information, see Use dynamic labels.
9. (Optional) To add a single metric to the graph, choose the **All metrics** tab and drill down to the metric you want.
10. (Optional) To change the time range shown on the graph, choose either **custom** at the top of the graph or one of the time periods to the left of **custom**.
11. (Optional) Horizontal annotations help dashboard users quickly see when a metric has spiked to a certain level or whether the metric is within a predefined range. To add a horizontal annotation, choose **Graph options** and then **Add horizontal annotation**:

    a. For **Label**, enter a label for the annotation.

b. For **Value**, enter the metric value where the horizontal annotation appears.

c. For **Fill**, specify whether to use fill shading with this annotation. For example, choose `Above` or `Below` for the corresponding area to be filled. If you specify `Between`, another `Value` field appears, and the area of the graph between the two values is filled.

d. For **Axis**, specify whether the numbers in `Value` refer to the metric associated with the left y-axis or the right y-axis if the graph includes multiple metrics.

You can change the fill color of an annotation by choosing the color square in the left column of the annotation.

Repeat these steps to add multiple horizontal annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

12. (Optional) Vertical annotations help you mark milestones in a graph, such as operational events or the beginning and end of a deployment. To add a vertical annotation, choose **Graph options** and then **Add vertical annotation**:

a. For **Label**, enter a label for the annotation. To show only the date and time on the annotation, keep the **Label** field blank.

b. For **Date**, specify the date and time where the vertical annotation appears.

c. For **Fill**, specify whether to use fill shading before or after a vertical annotation or between two vertical annotations. For example, choose `Before` or `After` for the corresponding area to be filled. If you specify `Between`, another `Date` field appears, and the area of the graph between the two values is filled.

Repeat these steps to add multiple vertical annotations to the same graph.

To hide an annotation, clear the check box in the left column for that annotation.

To delete an annotation, choose **x** in the **Actions** column.

13. Choose **Create widget**.

14. Choose **Save dashboard**.

To use the Metrics view to graph searched metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. In the search field, enter the tokens to search for: for example, **`cpuutilization t2.small`**.

   Results that match your search appear.
4. To graph all of the metrics that match your search, choose **Graph search**.

   or

   To refine your search, choose one of the namespaces that appeared in your search results.
5. If you selected a namespace to narrow your results, you can do the following:

   a. To graph one or more metrics, select the check box next to each metric. To select all metrics, select the check box in the heading row of the table.
   b. To refine your search, hover over a metric name and choose **Add to search** or **Search for this only**.
   c. To view help for a metric, select the metric name and choose **What is this?**.

   The selected metrics appear on the graph.
6. (Optional) Select one of the buttons in the search bar to edit that part of the search term.
7. (Optional) To add the graph to a dashboard, choose **Actions** and then **Add to dashboard**.

Use metric streams

You can use *metric streams* to continually stream CloudWatch metrics to a destination of your choice, with near-real-time delivery and low latency. Supported destinations include AWS destinations such as Amazon Simple Storage Service and several third-party service provider destinations.

There are three main usage scenarios for CloudWatch metric streams:

- **Custom setup with Firehose**— Create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream them to a data lake such as Amazon S3, or to any destination or endpoint supported by Firehose including third-party providers. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0

formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.

- **Quick S3 setup**— Stream to Amazon Simple Storage Service with a quick setup process. By default, CloudWatch creates the resources needed for the stream. JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported.
- **Quick AWS partner setup**— CloudWatch provides a quick setup experience for some third-party partners. You can use third-party service providers to monitor, troubleshoot, and analyze your applications using the streamed CloudWatch data. When you use the quick partner setup workflow, you need to provide only a destination URL and API key for your destination, and CloudWatch handles the rest of the setup. Quick partner setup is available for the following third-party providers:

  - Datadog
  - Dynatrace
  - Elastic
  - New Relic
  - Splunk Observability Cloud
  - SumoLogic

You can stream all of your CloudWatch metrics, or use filters to stream only specified metrics. Each metric stream can include up to 1000 filters that either include or exclude metric namespaces or specific metrics. A single metric stream can have only include or exclude filters, but not both.

After a metric stream is created, if new metrics are created that match the filters in place, the new metrics are automatically included in the stream.

There is no limit on the number of metric streams per account or per Region, and no limit on the number of metric updates being streamed.

Each stream can use either JSON format, OpenTelemetry 1.0.0, or OpenTelemetry 0.7.0 format. You can edit the output format of a metric stream at any time, such as for upgrading from OpenTelemetry 0.7.0 to OpenTelemetry 1.0.0. For more information about output formats, see CloudWatch metric stream output in JSON format, CloudWatch metric stream output in

OpenTelemetry 1.0.0 format, and CloudWatch metric stream output in OpenTelemetry 0.7.0 format.

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see CloudWatch cross-account observability.

Metric streams always include the `Minimum`, `Maximum`, `SampleCount`, and `Sum` statistics. You can also choose to include additional statistics at an additional charge. For more information, see Statistics that can be streamed.

Metric streams pricing is based on the number of metric updates. You also incur charges from Firehose for the delivery stream used for the metric stream. For more information, see Amazon CloudWatch Pricing.

Topics

- Set up a metric stream
- Statistics that can be streamed
- Metric stream operation and maintenance
- Monitoring your metric streams with CloudWatch metrics
- Trust between CloudWatch and Firehose
- CloudWatch metric stream output in JSON format
- CloudWatch metric stream output in OpenTelemetry 1.0.0 format
- CloudWatch metric stream output in OpenTelemetry 0.7.0 format
- Troubleshooting metric streams in CloudWatch

Set up a metric stream

Use the steps in the following sections to set up a CloudWatch metric stream.

After a metric stream is created, the time it takes for metric data to appear at the destination depends on the configured buffering settings on the Firehose delivery stream. The buffering is expressed in maximum payload size or maximum wait time, whichever is reached first. If these are set to the minimum values (60 seconds, 1MB) the expected latency is within 3 minutes if the selected CloudWatch namespaces have active metric updates.

In a CloudWatch metric stream, data is sent every minute. Data might arrive at the final destination out of order. All specified metrics in the specified namespaces are sent in the metric stream, except metrics with a timestamp that is more than two days old.

For each combination of metric name and namespace that you stream, all dimension combinations of that metric name and namespace are streamed.

For metric streams in monitoring accounts, you can choose whether to include metrics from the source accounts linked to that monitoring account. For more information, see CloudWatch cross-account observability.

To create and manage metric streams, you must be logged on to an account that has the **CloudWatchFullAccess** policy and the `iam:PassRole` permission, or an account that has the following list of permissions:

- `iam:PassRole`
- `cloudwatch:PutMetricStream`
- `cloudwatch:DeleteMetricStream`
- `cloudwatch:GetMetricStream`
- `cloudwatch:ListMetricStreams`
- `cloudwatch:StartMetricStreams`
- `cloudwatch:StopMetricStreams`

If you're going to have CloudWatch set up the IAM role needed for metric streams, you must also have the `iam:CreateRole` and `iam:PutRolePolicy` permissions.

> **Important**
>
> A user with the `cloudwatch:PutMetricStream` has access to the CloudWatch metric data that is being streamed, even if they don't have the `cloudwatch:GetMetricData` permission.

Topics

- Custom setup with Firehose
- Use Quick Amazon S3 setup
- Quick partner setup

Custom setup with Firehose

Use this method to create a metric stream and direct it to an Amazon Data Firehose delivery stream that delivers your CloudWatch metrics to where you want them to go. You can stream them to a data lake such as Amazon S3, or to any destination or endpoint supported by Firehose including third-party providers.

JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats are supported natively, or you can configure transformations in your Firehose delivery stream to convert the data to a different format such as Parquet. With a metric stream, you can continually update monitoring data, or combine this CloudWatch metric data with billing and performance data to create rich datasets. You can then use tools such as Amazon Athena to get insight into cost optimization, resource performance, and resource utilization.

You can use the CloudWatch console, the AWS CLI, AWS CloudFormation, or the AWS Cloud Development Kit (AWS CDK) to set up a metric stream.

The Firehose delivery stream that you use for your metric stream must be in the same account and the same Region where you set up the metric stream. To achieve cross-Region functionality, you can configure the Firehose delivery stream to stream to a final destination that is in a different account or a different Region.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream using Firehose.

To set up a custom metric stream using Firehose

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Custom setup with Firehose**.
5. For **Select your Kinesis Data Firehose stream**, select the Firehose delivery stream to use. It must be in the same account. The default format for this option is OpenTelemetry 0.7.0, but you can change the format later in this procedure.

Then select the Firehose delivery stream to use under **Select your Firehose delivery stream**.

6. (Optional)You can choose **Select existing service role** to use an existing IAM role instead of having CloudWatch create a new one for you.

   > **Note**
   >
   > If role manager is enabled in your account, CloudWatch attaches the role for you, and the role options described here (for example the **Create new role** button) are replaced by a **Customize** option. To use a different role, choose **Customize**. For more information, see IAM role creation in the *IAM User Guide*.

7. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.

8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

   If you choose **All metrics**, all metrics from this account will be included in the stream.

   Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

   If you choose **Select metrics**, do one of the following:

   - To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
   - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.

9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

For definitions of the additional statistics, see CloudWatch statistics definitions.

10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

AWS CLI or AWS API

Use the following steps to create a CloudWatch metric stream.

To use the AWS CLI or AWS API to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating a Firehose stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.
4. Use the `aws cloudwatch put-metric-stream` CLI command or the `PutMetricStream` API to create the CloudWatch metric stream.

AWS CloudFormation

You can use CloudFormation to set up a metric stream. For more information, see AWS::CloudWatch::MetricStream.

To use CloudFormation to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating a Firehose stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.

4. Create the stream in CloudFormation. For more information, see AWS::CloudWatch::MetricStream.

AWS Cloud Development Kit (AWS CDK)

You can use AWS Cloud Development Kit (AWS CDK) to set up a metric stream.

To use the AWS CDK to create a metric stream

1. If you're streaming to Amazon S3, first create the bucket. For more information, see Creating a bucket.
2. Create the Firehose delivery stream. For more information, see Creating an Amazon Data Firehose Delivery Stream.
3. Create an IAM role that enables CloudWatch to write to the Firehose delivery stream. For more information about the contents of this role, see Trust between CloudWatch and Firehose.
4. Create the metric stream. The metric stream resource is available in AWS CDK as a Level 1 (L1) Construct named `CfnMetricStream`. For more information, see Using L1 constructs.

Use Quick Amazon S3 setup

The **Quick S3 Setup** method works well if you want to quickly set up a stream to Amazon S3 and you don't need any formatting transformation beyond the supported JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0 formats. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format while you set up the stream.

Alternatively, if you want the final format to be Parquet format or Optimized Row Columnar (ORC), you should instead follow the steps in Custom setup with Firehose.

CloudWatch console

This section describes how to use the CloudWatch console to set up a metric stream Amazon S3 using Quick S3 setup.

To set up a metric stream using Quick S3 setup

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.

3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick S3 setup**. CloudWatch will create all necessary resources including the Firehose delivery stream and the necessary IAM roles. The default format for this option is JSON, but you can change the format later in this procedure.
5. (Optional) Choose **Select existing resources** to use an existing S3 bucket or existing IAM roles instead of having CloudWatch create new ones for you.

   > **Note**
   >
   > If role manager is enabled in your account, CloudWatch attaches the role for you, and the role options described here (for example the **Create new role** button) are replaced by a **Customize** option. To use a different role, choose **Customize**. For more information, see IAM role creation in the *IAM User Guide*.

6. (Optional) To change the output format from the default format for your scenario, choose **Change output format**. The supported formats are JSON, OpenTelemetry 1.0.0, and OpenTelemetry 0.7.0.
7. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

   If you choose **All metrics**, all metrics from this account will be included in the stream.

   Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

   If you choose **Select metrics**, do one of the following:

   - To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
   - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.

8. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

   To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

   Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

   For definitions of the additional statistics, see CloudWatch statistics definitions.
9. (Optional) Customize the name of the new metric stream under **Metric stream name**.
10. Choose **Create metric stream**.

Quick partner setup

CloudWatch provides a quick setup experience for the following third-party partners. To use this workflow, you need to provide only a destination URL and API key for your destination. CloudWatch handles the rest of setup including creating the Firehose delivery stream and the necessary IAM roles.

> **Important**
>
> Before you use quick partner setup to create a metric stream, we strongly recommend that you read that partner's documentation, linked in the following list.

- Datadog
- Dynatrace
- Elastic
- New Relic
- Splunk Observability Cloud
- SumoLogic

When you set up a metric stream to one of these partners, the stream is created with some default settings, as listed in the following sections.

Topics

- Set up a metric stream using quick partner setup
- Datadog stream defaults
- Dynatrace stream defaults
- Elastic stream defaults
- New Relic stream defaults
- Splunk Observability Cloud stream defaults
- Sumo Logic stream defaults

Set up a metric stream using quick partner setup

CloudWatch provides a quick setup option for some third-party partners. Before you start the steps in this section, you must have certain information for the partner. This information might include a destination URL and/or an API key for your partner destination. You should also read the documentation at the partner's website linked in the previous section, and the defaults for that partner listed in the following sections.

To stream to a third-party destination not supported by quick setup, you can follow the instructions in Follow the instructions in Custom setup with Firehose to set up a stream using Firehose, and then send those metrics from Firehose to the final destination.

To use quick partner setup to create a metric stream to third-party provider

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**. Then choose **Create metric stream**.
3. (Optional) If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can choose whether to include metrics from linked source accounts in this metric stream. To include metrics from source accounts, choose **Include source account metrics**.
4. Choose **Quick Amazon Web Services partner setup**
5. Select the name of the partner that you want to stream metrics to.
6. For **Endpoint URL**, enter the destination URL.

7. For **Access Key** or **API Key**, enter the access key for the partner. Not all partners require an access key.
8. For **Metrics to be streamed**, choose either **All metrics** or **Select metrics**.

   If you choose **All metrics**, all metrics from this account will be included in the stream.

   Consider carefully whether to stream all metrics, because the more metrics that you stream the higher your metric stream charges will be.

   If you choose **Select metrics**, do one of the following:

   - To stream most metric namespaces, choose **Exclude** and select the namespaces or metrics to exclude. When you specify a namespace in **Exclude**, you can optionally select some specific metrics from that namespace to exclude. If you choose to exclude a namespace but don't then select metrics in that namespace, all metrics from that namespace are excluded.
   - To include only a few metric namespaces or metrics in the metric stream, choose **Include** and then select the namespaces or metrics to include. If you choose to include a namespace but don't then select metrics in that namespace, all metrics from that namespace are included.
9. (Optional) To stream additional statistics for some of these metrics beyond Minimum, Maximum, SampleCount, and Sum, choose **Add additional statistics**. Either choose **Add recommended metrics** to add some commonly used statistics, or manually select the namespace and metric name to stream additional statistics for. Next, select the additional statistics to stream.

   To then choose another group of metrics to stream a different set of additional statistics for, choose **Add additional statistics**. Each metric can include as many as 20 additional statistics, and as many as 100 metrics within a metric stream can include additional statistics.

   Streaming additional statistics incurs more charges. For more information, see Statistics that can be streamed.

   For definitions of the additional statistics, see CloudWatch statistics definitions.
10. (Optional) Customize the name of the new metric stream under **Metric stream name**.
11. Choose **Create metric stream**.

Datadog stream defaults

Quick partner setup streams to Datadog use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 4 MBs
- **Firehose stream retry option** Duration of 60 seconds

When you use quick partner setup to create a metric stream to Datadog and you stream certain metrics, by default those metrics include some additional statistics. Streaming additional statistics can incur additional charges. For more information about statistics and their charges, see Statistics that can be streamed.

The following list shows the metrics that have additional statistics streamed by default, if you choose to stream those metrics. You can choose to de-select these additional statistics before you start the stream.

- **`Duration` in `AWS/Lambda`:** p50, p80, p95, p99, p99.9
- **`PostRuntimeExtensionDuration` in `AWS/Lambda`:** p50, p99
- **`FirstByteLatency` and `TotalRequestLatency`in `AWS/S3`:** p50, p90, p95, p99, p99.9
- **`ResponseLatency` in `AWS/Polly` and `TargetResponseTime` in `AWS/ApplicationELB`:** p50, p90, p95, p99
- **`Latency` and `IntegrationLatency` in `AWS/ApiGateway`:** p90, p95, p99
- **`Latency` and `TargetResponseTime` in `AWS/ELB`:** p95, p99
- **`RequestLatency` in `AWS/AppRunner`:** p50, p95, p99
- **`ActivityTime`, `ExecutionTime`, `LambdaFunctionRunTime`, `LambdaFunctionScheduleTime`, `LambdaFunctionTime`, `ActivityRunTime`, and `ActivityScheduleTime` in `AWS/States`:** p95, p99
- **`EncoderBitRate`, `ConfiguredBitRate`, and `ConfiguredBitRateAvailable` in `AWS/MediaLive`:** p90
- **`Latency` in `AWS/AppSync`:** p90

Dynatrace stream defaults

Quick partner setup streams to Dynatrace use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 5 MBs
- **Firehose stream retry option** Duration of 600 seconds

Elastic stream defaults

Quick partner setup streams to Elastic use the following defaults:

- **Output format:** OpenTelemetry 1.0.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

New Relic stream defaults

Quick partner setup streams to New Relic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Splunk Observability Cloud stream defaults

Quick partner setup streams to Splunk Observability Cloud use the following defaults:

- **Output format:** OpenTelemetry 1.0.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 300 seconds

Sumo Logic stream defaults

Quick partner setup streams to Sumo Logic use the following defaults:

- **Output format:** OpenTelemetry 0.7.0
- **Firehose stream content encoding** GZIP
- **Firehose stream buffering options** Interval of 60 seconds, size of 1 MB
- **Firehose stream retry option** Duration of 60 seconds

Statistics that can be streamed

Metric streams always include the following statistics: `Minimum`, `Maximum`, `SampleCount`, and `Sum`. You can also choose to include the following additional statistics in a metric stream. This choice is on a per-metric basis. For more information about these statistics, see CloudWatch statistics definitions.

- Percentile values such as p95 or p99 (For streams with either JSON or OpenTelemetry format)
- Trimmed mean (Only for streams with the JSON format)
- Winsorized mean (Only for streams with the JSON format)
- Trimmed count (Only for streams with the JSON format)
- Trimmed sum (Only for streams with the JSON format)
- Percentile rank (Only for streams with the JSON format)
- Interquartile mean (Only for streams with the JSON format)

Streaming additional statistics incurs additional charges. Streaming between one and five of these additional statistics for a particular metric is billed as an additional metric update. Thereafter, each additional set of up to five of these statistics is billed as another metric update.

For example, suppose that for one metric you are streaming the following six additional statistics: p95, p99, p99.9, Trimmed mean, Winsorized mean, and Trimmed sum. Each update of this metric is billed as three metric updates: one for the metric update which includes the default statistics, one for the first five additional statistics, and one for the sixth additional statistic. Adding up to four more additional statistics for a total of ten would not increase the billing, but an eleventh additional statistic would do so.

When you specify a metric name and namespace combination to stream additional statistics, all dimension combinations of that metric name and namespace are streamed with the additional statistics.

CloudWatch metric streams publishes a new metric, `TotalMetricUpdate`, which reflects the base number of metric updates plus extra metric updates incurred by streaming additional statistics. For more information, see Monitoring your metric streams with CloudWatch metrics.

For more information, see Amazon CloudWatch Pricing.

> **Note**
>
> Some metrics do not support percentiles. Percentile statistics for these metrics are excluded from the stream and do not incur metric stream charges. An example of these statistics that do not support percentiles are some metrics in the `AWS/ECS` namespace.

The additional statistics that you configure are streamed only if they match the filters for the stream. For example, if you create a stream that has only `EC2` and `RDS` in the include filters, and then your statistics configuration lists `EC2` and `Lambda`, then the stream includes `EC2` metrics with additional statistics, `RDS` metrics with only the default statistics, and doesn't include `Lambda` statistics at all.

Metric stream operation and maintenance

Metric streams are always in one of two states, **Running** or **Stopped**.

- **Running**— The metric stream is running correctly. There might not be any metric data streamed to the destination because of the filters on the stream.
- **Stopped**— The metric stream has been explicitly stopped by someone, and not because of an error. It might be useful to stop your stream to temporarily pause the streaming of data without deleting the stream.

If you stop and restart a metric stream, the metric data that was published to CloudWatch while the metric stream was stopped is not backfilled to the metric stream.

If you change the output format of a metric stream, in certain cases you might see a small amount of metric data written to the destination in both the old format and the new format. To avoid

this situation, you can create a new Firehose delivery stream with the same configuration as your current one, then change to the new Firehose delivery stream and change the output format at the same time. This way, the Kinesis records with different output format are stored on Amazon S3 in separate objects. Later, you can direct the traffic back to the original Firehose delivery stream and delete the second delivery stream.

To view, edit, stop, and start your metric streams

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **Streams**.

   The list of streams appears, and the **Status** column displays whether each stream is running or stopped.
3. To stop or start a metric stream, select the stream and choose **Stop** or **Start**.
4. To see the details about a metric stream, select the stream and choose **View details**.
5. To change the stream's output format, filters, destination Firehose stream, or roles, choose **Edit** and make the changes that you want.

   If you change the filters, there might be some gaps in the metric data during the transition.

Monitoring your metric streams with CloudWatch metrics

Metric streams emit CloudWatch metrics about their health and operation in the `AWS/CloudWatch/MetricStreams` namespace. The following metrics are emitted. These metrics are emitted with a `MetricStreamName` dimension and with no dimension. You can use the metrics with no dimensions to see aggregated metrics for all of your metric streams. You can use the metrics with the `MetricStreamName` dimension to see the metrics about only that metric stream.

For all of these metrics, values are emitted only for metric streams that are in the **Running** state.

Metric	Description
`MetricUpdate`	The number of metric updates sent to the metric stream. If no metric updates are streamed during a time period, this metric is not emitted during that time period. If you stop the metric stream, this metric stops being emitted until the metric stream is started again.

Metric	Description
	Valid Statistic: Sum Units: None
TotalMetricUpdate	This is calculated as **MetricUpdate + a number based on additional statistics that are being streamed**. For each unique namespace and metric name combination, streaming 1-5 additional statistics adds 1 to the TotalMetricUpdate , streaming 6-10 additional statistics adds 2 to TotalMetricUpdate , and so on. Valid Statistic: Sum Units: None
PublishErrorRate	The number of unrecoverable errors that occur when putting data into the Firehose delivery stream. If no errors occur during a time period, this metric is not emitted during that time period. If you stop the metric stream, this metric stops being emitted until the metric stream is started again. Valid Statistic: Average to see the rate of metric updates unable to be written. This value will be between 0.0 and 1.0. Units: None

Trust between CloudWatch and Firehose

The Firehose delivery stream must trust CloudWatch through an IAM role that has write permissions to Firehose. These permissions can be limited to the single Firehose delivery stream that the CloudWatch metric stream uses. The IAM role must trust the streams.metrics.cloudwatch.amazonaws.com service principal.

If you use the CloudWatch console to create a metric stream, you can have CloudWatch create the role with the correct permissions. If you use another method to create a metric stream, or you want to create the IAM role itself, it must contain the following permissions policy and trust policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:firehose:us-east-1:123456789012:deliverystream/
*"
        }
    ]
}
```

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "streams.metrics.cloudwatch.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Metric data is streamed by CloudWatch to the destination Firehose delivery stream on behalf of the source that owns the metric stream resource.

CloudWatch metric stream output in JSON format

In a CloudWatch metric stream that uses the JSON format, each Firehose record contains multiple JSON objects separated by a newline character (\n). Each object includes a single data point of a single metric.

The JSON format that is used is fully compatible with AWS Glue and with Amazon Athena. If you have a Firehose delivery stream and an AWS Glue table formatted correctly, the format can be automatically transformed into Parquet format or Optimized Row Columnar (ORC) format before being stored in S3. For more information about transforming the format, see Converting Your Input Record Format in Firehose. For more information about the correct format for AWS Glue, see Which AWS Glue schema should I use for JSON output format?.

In the JSON format, the valid values for `unit` are the same as for the value of `unit` in the `MetricDatum` API structure. For more information, see MetricDatum. The value for the `timestamp` field is in epoch milliseconds, such as `1616004674229`.

The following is an example of the format. In this example, the JSON is formatted for easy reading, but in practice the whole format is on a single line.

```
{
    "metric_stream_name": "MyMetricStream",
    "account_id": "1234567890",
    "region": "us-east-1",
    "namespace": "AWS/EC2",
    "metric_name": "DiskWriteOps",
    "dimensions": {
        "InstanceId": "i-123456789012"
    },
    "timestamp": 1611929698000,
    "value": {
        "count": 3.0,
        "sum": 20.0,
        "max": 18.0,
        "min": 0.0,
        "p99": 17.56,
        "p99.9": 17.8764,
        "TM(25%:75%)": 16.43
    },
    "unit": "Seconds"
}
```

Which AWS Glue schema should I use for JSON output format?

The following is an example of a JSON representation of the `StorageDescriptor` for an AWS Glue table, which would then be used by Firehose. For more information about `StorageDescriptor`, see StorageDescriptor.

```
{
  "Columns": [
    {
      "Name": "metric_stream_name",
      "Type": "string"
    },
    {
      "Name": "account_id",
      "Type": "string"
    },
    {
      "Name": "region",
      "Type": "string"
    },
    {
      "Name": "namespace",
      "Type": "string"
    },
    {
      "Name": "metric_name",
      "Type": "string"
    },
    {
      "Name": "timestamp",
      "Type": "timestamp"
    },
    {
      "Name": "dimensions",
      "Type": "map<string,string>"
    },
    {
      "Name": "value",
      "Type":
 "struct<min:double,max:double,count:double,sum:double,p99:double,p99.9:double>"
    },
    {
      "Name": "unit",
```

```
      "Type": "string"
    }
  ],
  "Location": "s3://amzn-s3-demo-bucket/",
  "InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
  "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
  "SerdeInfo": {
    "SerializationLibrary": "org.apache.hive.hcatalog.data.JsonSerDe"
  },
  "Parameters": {
    "classification": "json"
  }
}
```

The preceding example is for data written on Amazon S3 in JSON format. Replace the values in the following fields with the indicated values to store the data in Parquet format or Optimized Row Columnar (ORC) format.

- **Parquet:**
  - inputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
  - parameters.classification: parquet
- **ORC:**
  - inputFormat: org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
  - outputFormat: org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
  - SerDeInfo.serializationLib: org.apache.hadoop.hive.ql.io.orc.OrcSerde
  - parameters.classification: orc

CloudWatch metric stream output in OpenTelemetry 1.0.0 format

Note

With the OpenTelemetry 1.0.0 format, metric attributes are encoded as a list of `KeyValue` objects instead of the `StringKeyValue` type used in the 0.7.0 format. As a consumer, this is the only major change between the 0.7.0 and 1.0.0 formats. A parser generated from the 0.7.0 proto files won't parse metric attributes encoded in the 1.0.0 format. The same is

true in reverse, a parser generated from the 1.0.0 proto files will not parse metric attributes encoded in the 0.7.0 format.

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see OpenTelemetry.

For information about the full OpenTelemetry 1.0.0 specification, see Release version 1.0.0.

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {
  resource {
    attributes {
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "123456789012"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
```

```
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  scope_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "NoneTranslated"
      summary {
        data_points {
          start_time_unix_nano: 60000000000
          time_unix_nano: 120000000000
          count: 1
          sum: 1.0
          quantile_values {
            value: 1.0
          }
          quantile_values {
            quantile: 0.95
            value: 1.0
          }
          quantile_values {
            quantile: 0.99
            value: 1.0
          }
          quantile_values {
            quantile: 1.0
            value: 1.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
            }
          }
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
            }
          }
          attributes {
            key: "Dimensions"
```

```
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
                  }
                }
              }
            }
          }
        }
        data_points {
          start_time_unix_nano: 70000000000
          time_unix_nano: 130000000000
          count: 2
          sum: 5.0
          quantile_values {
            value: 2.0
          }
          quantile_values {
            quantile: 1.0
            value: 3.0
          }
          attributes {
            key: "Namespace"
            value {
              string_value: "AWS/DynamoDB"
            }
          }
          attributes {
            key: "MetricName"
            value {
              string_value: "ConsumedReadCapacityUnits"
            }
          }
          attributes {
            key: "Dimensions"
            value {
              kvlist_value {
                values {
                  key: "TableName"
                  value {
                    string_value: "MyTable"
```

```
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` is the top-level object to represent MetricData objects.

```
// A collection of ScopeMetrics from a Resource.
message ResourceMetrics {
  reserved 1000;

  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated ScopeMetrics scope_metrics = 2;

  // This schema_url applies to the data in the "resource" field. It does not apply
  // to the data in the "scope_metrics" field which have their own schema_url field.
  string schema_url = 3;
}
```

The Resource object

A `Resource` object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by AWS, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The `Resource` object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of attributes that describe the resource.
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
 then
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

The ScopeMetrics object

The `scope` field will not be filled. We fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an Scope.
message ScopeMetrics {
  // The instrumentation scope information for the metrics in this message.
  // Semantically when InstrumentationScope isn't set, it is equivalent with
  // an empty instrumentation scope name (unknown).
  opentelemetry.proto.common.v1.InstrumentationScope scope = 1;

  // A list of metrics that originate from an instrumentation library.
```

```
  repeated Metric metrics = 2;

  // This schema_url applies to all metrics in the "metrics" field.
  string schema_url = 3;
}
```

The Metric object

The metric object contains some metadata and a `Summary` data field that contains a list of `SummaryDataPoint`.

For metric streams, the metadata is as follows:

- name will be `amazonaws.com/`*`metric_namespace`*`/`*`metric_name`*
- `description` will be blank
- `unit` will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see Translations with OpenTelemetry 1.0.0 format in CloudWatch and The Unified Code For Units of Measure.
- `type` will be `SUMMARY`

```
message Metric {
  reserved 4, 6, 8;

  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;

  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  // Data determines the aggregation type (if any) of the metric, what is the
  // reported value type for the data points, as well as the relatationship to
  // the time interval over which they are reported.
  oneof data {
    Gauge gauge = 5;
    Sum sum = 7;
    Histogram histogram = 9;
```

```
    ExponentialHistogram exponential_histogram = 10;
    Summary summary = 11;
  }
}

message Summary {
  repeated SummaryDataPoint data_points = 1;
}
```

The SummaryDataPoint object

The SummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// SummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message SummaryDataPoint {
  reserved 1;

  // The set of key/value pairs that uniquely identify the timeseries from
  // where this point belongs. The list may be empty (may contain 0 elements).
  // Attribute keys MUST be unique (it is not allowed to have more than one
  // attribute with the same key).
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 7;

  // StartTimeUnixNano is optional but strongly encouraged, see the
  // the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 start_time_unix_nano = 2;

  // TimeUnixNano is required, see the detailed comments above Metric.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
```

```
  //
  // Note: Sum should only be filled out when measuring non-negative discrete
  // events, and is assumed to be monotonic over the values of these events.
  // Negative events *can* be recorded, but sum should not be filled out when
  // doing so.  This is specifically to enforce compatibility w/ OpenMetrics,
  // see: https://github.com/OpenObservability/OpenMetrics/blob/main/specification/
OpenMetrics.md#summary
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  //
  // See the following issue for more context:
  // https://github.com/open-telemetry/opentelemetry-proto/issues/125
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    //
    // Quantile values must NOT be negative.
    double value = 2;
  }

  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;

  // Flags that apply to this specific data point.  See DataPointFlags
  // for the available flags and their meaning.
  uint32 flags = 8;
}
```

For more information, see [Translations with OpenTelemetry 1.0.0 format in CloudWatch].

Translations with OpenTelemetry 1.0.0 format in CloudWatch

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One attribute has the key `Namespace` and its value is the namespace of the metric
- One attribute has the key `MetricName` and its value is the name of the metric
- One pair has the key `Dimensions` and its value is a nested list of key-value pairs. Each pair in this list maps to a CloudWatch metric dimension, where the pair's key is the name of the dimension and its value is the value of the dimension.

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
  - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see The Unified Code For Units of Measure.

CloudWatch	OpenTelemetry
Second	s
Second or Seconds	s

CloudWatch	OpenTelemetry
Microseconds	us
Milliseconds	ms
Bytes	By
Kilobytes	kBy
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
Percent	%
Count	{Count}
None	1

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 1.0.0 messages

This section provides information to help you get started with parsing OpenTelemetry 1.0.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 1.0.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
  - To use Java, add the following Maven dependency to your Java project: OpenTelemetry Java >> 0.14.1.
  - To use any other language, follow these steps:

    a. Make sure that your language is supported by checking the list at Generating Your Classes.

    b. Install the Protobuf compiler by following the steps at Download Protocol Buffers.

    c. Download the OpenTelemetry 1.0.0 ProtoBuf definitions at Release version 1.0.0.

    d. Confirm that you are in the root folder of the downloaded OpenTelemetry 1.0.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see Generating Your Classes.

    The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {
```

```
    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array.number` 8-bit integers

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
```

```
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
```

```
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see Download Protocol Buffers.

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

CloudWatch metric stream output in OpenTelemetry 0.7.0 format

OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis. OpenTelemetry is part of the Cloud Native Computing Foundation. For more information, see OpenTelemetry.

For information about the full OpenTelemetry 0.7.0 specification, see v0.7.0 release.

A Kinesis record can contain one or more `ExportMetricsServiceRequest` OpenTelemetry data structures. Each data structure starts with a header with an `UnsignedVarInt32` indicating the record length in bytes. Each `ExportMetricsServiceRequest` may contain data from multiple metrics at once.

The following is a string representation of the message of the `ExportMetricsServiceRequest` OpenTelemetry data structure. OpenTelemetry serializes the Google Protocol Buffers binary protocol, and this is not human-readable.

```
resource_metrics {
  resource {
    attributes {
```

```
      key: "cloud.provider"
      value {
        string_value: "aws"
      }
    }
    attributes {
      key: "cloud.account.id"
      value {
        string_value: "2345678901"
      }
    }
    attributes {
      key: "cloud.region"
      value {
        string_value: "us-east-1"
      }
    }
    attributes {
      key: "aws.exporter.arn"
      value {
        string_value: "arn:aws:cloudwatch:us-east-1:123456789012:metric-stream/
MyMetricStream"
      }
    }
  }
  instrumentation_library_metrics {
    metrics {
      name: "amazonaws.com/AWS/DynamoDB/ConsumedReadCapacityUnits"
      unit: "1"
      double_summary {
        data_points {
          labels {
            key: "Namespace"
            value: "AWS/DynamoDB"
          }
          labels {
            key: "MetricName"
            value: "ConsumedReadCapacityUnits"
          }
          labels {
            key: "TableName"
            value: "MyTable"
          }
          start_time_unix_nano: 1604948400000000000
```

```
      time_unix_nano: 1604948460000000000
      count: 1
      sum: 1.0
      quantile_values {
        quantile: 0.0
        value: 1.0
      }
      quantile_values {
        quantile: 0.95
        value: 1.0
      }
      quantile_values {
        quantile: 0.99
        value: 1.0
      }
      quantile_values {
        quantile: 1.0
        value: 1.0
      }
    }
    data_points {
      labels {
        key: "Namespace"
        value: "AWS/DynamoDB"
      }
      labels {
        key: "MetricName"
        value: "ConsumedReadCapacityUnits"
      }
      labels {
        key: "TableName"
        value: "MyTable"
      }
      start_time_unix_nano: 1604948460000000000
      time_unix_nano: 1604948520000000000
      count: 2
      sum: 5.0
      quantile_values {
        quantile: 0.0
        value: 2.0
      }
      quantile_values {
        quantile: 1.0
        value: 3.0
```

```
          }
        }
      }
    }
  }
}
```

Top-level object to serialize OpenTelemetry metric data

`ExportMetricsServiceRequest` is the top-level wrapper to serialize an OpenTelemetry exporter payload. It contains one or more `ResourceMetrics`.

```
message ExportMetricsServiceRequest {
  // An array of ResourceMetrics.
  // For data coming from a single resource this array will typically contain one
  // element. Intermediary nodes (such as OpenTelemetry Collector) that receive
  // data from multiple origins typically batch the data before forwarding further and
  // in that case this array will contain multiple elements.
  repeated opentelemetry.proto.metrics.v1.ResourceMetrics resource_metrics = 1;
}
```

`ResourceMetrics` is the top-level object to represent MetricData objects.

```
// A collection of InstrumentationLibraryMetrics from a Resource.
message ResourceMetrics {
  // The resource for the metrics in this message.
  // If this field is not set then no resource info is known.
  opentelemetry.proto.resource.v1.Resource resource = 1;

  // A list of metrics that originate from a resource.
  repeated InstrumentationLibraryMetrics instrumentation_library_metrics = 2;
}
```

The Resource object

A `Resource` object is a value-pair object that contains some information about the resource that generated the metrics. For metrics created by AWS, the data structure contains the Amazon Resource Name (ARN) of the resource related to the metric, such as an EC2 instance or an S3 bucket.

The `Resource` object contains an attribute called `attributes`, which store a list of key-value pairs.

- `cloud.account.id` contains the account ID
- `cloud.region` contains the Region
- `aws.exporter.arn` contains the metric stream ARN
- `cloud.provider` is always `aws`.

```
// Resource information.
message Resource {
  // Set of labels that describe the resource.
  repeated opentelemetry.proto.common.v1.KeyValue attributes = 1;

  // dropped_attributes_count is the number of dropped attributes. If the value is 0,
  // no attributes were dropped.
  uint32 dropped_attributes_count = 2;
}
```

The InstrumentationLibraryMetrics object

The instrumentation_library field will not be filled. We will fill only the metrics field that we are exporting.

```
// A collection of Metrics produced by an InstrumentationLibrary.
message InstrumentationLibraryMetrics {
  // The instrumentation library information for the metrics in this message.
  // If this field is not set then no library info is known.
  opentelemetry.proto.common.v1.InstrumentationLibrary instrumentation_library = 1;
  // A list of metrics that originate from an instrumentation library.
  repeated Metric metrics = 2;
}
```

The Metric object

The metric object contains a `DoubleSummary` data field that contains a list of `DoubleSummaryDataPoint`.

```
message Metric {
  // name of the metric, including its DNS name prefix. It must be unique.
  string name = 1;

  // description of the metric, which can be used in documentation.
  string description = 2;
```

```
  // unit in which the metric value is reported. Follows the format
  // described by http://unitsofmeasure.org/ucum.html.
  string unit = 3;

  oneof data {
    IntGauge int_gauge = 4;
    DoubleGauge double_gauge = 5;
    IntSum int_sum = 6;
    DoubleSum double_sum = 7;
    IntHistogram int_histogram = 8;
    DoubleHistogram double_histogram = 9;
    DoubleSummary double_summary = 11;
  }
}

message DoubleSummary {
  repeated DoubleSummaryDataPoint data_points = 1;
}
```

The MetricDescriptor object

The MetricDescriptor object contains metadata. For more information, see metrics.proto on GitHub.

For metric streams, the MetricDescriptor has the following contents:

- name will be amazonaws.com/*metric_namespace*/*metric_name*
- `description` will be blank.
- `unit` will be filled by mapping the metric datum's unit to the case-sensitive variant of the Unified code for Units of Measure. For more information, see Translations with OpenTelemetry 0.7.0 format in CloudWatch and The Unified Code For Units of Measure.
- `type` will be SUMMARY.

The DoubleSummaryDataPoint object

The DoubleSummaryDataPoint object contains the value of a single data point in a time series in a DoubleSummary metric.

```
// DoubleSummaryDataPoint is a single data point in a timeseries that describes the
// time-varying values of a Summary metric.
message DoubleSummaryDataPoint {
```

```
  // The set of labels that uniquely identify this timeseries.
  repeated opentelemetry.proto.common.v1.StringKeyValue labels = 1;

  // start_time_unix_nano is the last time when the aggregation value was reset
  // to "zero". For some metric types this is ignored, see data types for more
  // details.
  //
  // The aggregation value is over the time interval (start_time_unix_nano,
  // time_unix_nano].
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  //
  // Value of 0 indicates that the timestamp is unspecified. In that case the
  // timestamp may be decided by the backend.
  fixed64 start_time_unix_nano = 2;

  // time_unix_nano is the moment when this aggregation value was reported.
  //
  // Value is UNIX Epoch time in nanoseconds since 00:00:00 UTC on 1 January
  // 1970.
  fixed64 time_unix_nano = 3;

  // count is the number of values in the population. Must be non-negative.
  fixed64 count = 4;

  // sum of the values in the population. If count is zero then this field
  // must be zero.
  double sum = 5;

  // Represents the value at a given quantile of a distribution.
  //
  // To record Min and Max values following conventions are used:
  // - The 1.0 quantile is equivalent to the maximum value observed.
  // - The 0.0 quantile is equivalent to the minimum value observed.
  message ValueAtQuantile {
    // The quantile of a distribution. Must be in the interval
    // [0.0, 1.0].
    double quantile = 1;

    // The value at the given quantile of a distribution.
    double value = 2;
  }
```

```
  // (Optional) list of values at different quantiles of the distribution calculated
  // from the current snapshot. The quantiles must be strictly increasing.
  repeated ValueAtQuantile quantile_values = 6;
}
```

For more information, see Translations with OpenTelemetry 0.7.0 format in CloudWatch.

Translations with OpenTelemetry 0.7.0 format in CloudWatch

CloudWatch performs some transformations to put CloudWatch data into OpenTelemetry format.

Translating namespace, metric name, and dimensions

These attributes are key-value pairs encoded in the mapping.

- One pair contains the namespace of the metric
- One pair contains the name of the metric
- For each dimension, CloudWatch stores the following pair: `metricDatum.Dimensions[i].Name, metricDatum.Dimensions[i].Value`

Translating Average, Sum, SampleCount, Min and Max

The Summary datapoint enables CloudWatch to export all of these statistics using one datapoint.

- `startTimeUnixNano` contains the CloudWatch `startTime`
- `timeUnixNano` contains the CloudWatch `endTime`
- `sum` contains the Sum statistic.
- `count` contains the SampleCount statistic.
- `quantile_values` contains two `valueAtQuantile.value` objects:
  - `valueAtQuantile.quantile = 0.0` with `valueAtQuantile.value = Min value`
  - `valueAtQuantile.quantile = 0.99` with `valueAtQuantile.value = p99 value`
  - `valueAtQuantile.quantile = 0.999` with `valueAtQuantile.value = p99.9 value`
  - `valueAtQuantile.quantile = 1.0` with `valueAtQuantile.value = Max value`

Resources that consume the metric stream can calculate the Average statistic as **Sum/SampleCount**.

Translating units

CloudWatch units are mapped to the case-sensitive variant of the Unified code for Units of Measure, as shown in the following table. For more information, see The Unified Code For Units of Measure.

CloudWatch	OpenTelemetry
Second	s
Second or Seconds	s
Microsecond	us
Milliseconds	ms
Bytes	By
Kilobytes	kBy
Megabytes	MBy
Gigabytes	GBy
Terabytes	TBy
Bits	bit
Kilobits	kbit
Megabits	MBit
Gigabits	GBit
Terabits	Tbit
Percent	%
Count	{Count}
None	1

Units that are combined with a slash are mapped by applying the OpenTelemetry conversion of both the units. For example, Bytes/Second is mapped to By/s.

How to parse OpenTelemetry 0.7.0 messages

This section provides information to help you get started with parsing OpenTelemetry 0.7.0.

First, you should get language-specific bindings, which enable you to parse OpenTelemetry 0.7.0 messages in your preferred language.

To get language-specific bindings

- The steps depend on your preferred language.
  - To use Java, add the following Maven dependency to your Java project: OpenTelemetry Java >> 0.14.1.
  - To use any other language, follow these steps:

    a. Make sure that your language is supported by checking the list at Generating Your Classes.

    b. Install the Protobuf compiler by following the steps at Download Protocol Buffers.

    c. Download the OpenTelemetry 0.7.0 ProtoBuf definitions at v0.7.0 release.

    d. Confirm that you are in the root folder of the downloaded OpenTelemetry 0.7.0 ProtoBuf definitions. Then create a `src` folder and then run the command to generate language-specific bindings. For more information, see Generating Your Classes.

    The following is an example for how to generate Javascript bindings.

```
protoc --proto_path=./ --js_out=import_style=commonjs,binary:src \
opentelemetry/proto/common/v1/common.proto \
opentelemetry/proto/resource/v1/resource.proto \
opentelemetry/proto/metrics/v1/metrics.proto \
opentelemetry/proto/collector/metrics/v1/metrics_service.proto
```

The following section includes examples of using the language-specific bindings that you can build using the previous instructions.

Java

```
package com.example;

import io.opentelemetry.proto.collector.metrics.v1.ExportMetricsServiceRequest;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class MyOpenTelemetryParser {

    public List<ExportMetricsServiceRequest> parse(InputStream inputStream) throws
 IOException {
        List<ExportMetricsServiceRequest> result = new ArrayList<>();

        ExportMetricsServiceRequest request;
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|pExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        while ((request =
 ExportMetricsServiceRequest.parseDelimitedFrom(inputStream)) != null) {
            // Do whatever we want with the parsed message
            result.add(request);
        }

        return result;
    }
}
```

Javascript

This example assumes that the root folder with the bindings generated is `./`

The data argument of the function `parseRecord` can be one of the following types:

- `Uint8Array` this is optimal
- `Buffer` optimal under node
- `Array.number` 8-bit integers

```
const pb = require('google-protobuf')
const pbMetrics =
    require('./opentelemetry/proto/collector/metrics/v1/metrics_service_pb')

function parseRecord(data) {
    const result = []

    // Loop until we've read all the data from the buffer
    while (data.length) {
        /* A Kinesis record can contain multiple `ExportMetricsServiceRequest`
           records, each of them starting with a header with an
           UnsignedVarInt32 indicating the record length in bytes:
            ------ -------------------------- ------ ----------------------
           |UINT32|ExportMetricsServiceRequest|UINT32|ExportMetricsService...
            ------ -------------------------- ------ ----------------------
         */
        const reader = new pb.BinaryReader(data)
        const messageLength = reader.decoder_.readUnsignedVarint32()
        const messageFrom = reader.decoder_.cursor_
        const messageTo = messageFrom + messageLength

        // Extract the current `ExportMetricsServiceRequest` message to parse
        const message = data.subarray(messageFrom, messageTo)

        // Parse the current message using the ProtoBuf library
        const parsed =
            pbMetrics.ExportMetricsServiceRequest.deserializeBinary(message)

        // Do whatever we want with the parsed message
        result.push(parsed.toObject())

        // Shrink the remaining buffer, removing the already parsed data
        data = data.subarray(messageTo)
    }

    return result
}
```

Python

You must read the `var-int` delimiters yourself or use the internal methods `_VarintBytes(size)` and `_DecodeVarint32(buffer, position)`. These return the position

in the buffer just after the size bytes. The read-side constructs a new buffer that is limited to reading only the bytes of the message.

```
size = my_metric.ByteSize()
f.write(_VarintBytes(size))
f.write(my_metric.SerializeToString())
msg_len, new_pos = _DecodeVarint32(buf, 0)
msg_buf = buf[new_pos:new_pos+msg_len]
request = metrics_service_pb.ExportMetricsServiceRequest()
request.ParseFromString(msg_buf)
```

Go

Use `Buffer.DecodeMessage()`.

C#

Use `CodedInputStream`. This class can read size-delimited messages.

C++

The functions described in `google/protobuf/util/delimited_message_util.h` can read size-delimited messages.

Other languages

For other languages, see Download Protocol Buffers.

When implementing the parser, consider that a Kinesis record can contain multiple `ExportMetricsServiceRequest` Protocol Buffers messages, each of them starting with a header with an `UnsignedVarInt32` that indicates the record length in bytes.

Troubleshooting metric streams in CloudWatch

If you're not seeing metric data at your final destination, check the following:

- Check that the metric stream is in the running state. For steps on how to use the CloudWatch console to do this, see Metric stream operation and maintenance.
- Metrics published more than two days in the past are not streamed. To determine whether a particular metric will be streamed, graph the metric in the CloudWatch console and check how

old the last visible datapoint is. If it is more than two days in the past, then it won't be picked up by metric streams.

- Check the metrics emitted by the metric stream. In the CloudWatch console, under **Metrics**, look at the **AWS/CloudWatch/MetricStreams** namespace for the **MetricUpdate**, **TotalMetricUpdate**, and **PublishErrorRate** metrics.
- If the **PublishErrorRate** metric is high, confirm that the destination that is used by the Firehose delivery stream exists and that the IAM role specified in the metric stream's configuration grants the `CloudWatch` service principal permissions to write to it. For more information, see Trust between CloudWatch and Firehose.
- Check that the Firehose delivery stream has permission to write to the final destination.
- In the Firehose console, view the Firehose delivery stream that is used for the metric stream and check the **Monitoring** tab to see whether the Firehose delivery stream is receiving data.
- Confirm that you have configured your Firehose delivery stream with the correct details.
- Check any available logs or metrics for the final destination that the Firehose delivery stream writes to.
- Ensure that CloudWatch Logs decompression is disabled on your Firehose delivery stream. Only requests from CloudWatch Logs are supported when decompression is enabled.
- To get more detailed information, enable CloudWatch Logs error logging on the Firehose delivery stream. For more information, see Monitoring Amazon Data Firehose Using CloudWatch Logs.

> **Note**
>
> After a data point for a specific metric and timestamp has been sent, it won't be sent again even if the value of the data point changes later.

Math expressions with metrics

Metric math enables you to query multiple CloudWatch metrics and use math expressions to create new time series based on these metrics. You can visualize the resulting time series on the CloudWatch console and add them to dashboards. Using AWS Lambda metrics as an example, you could divide the `Errors` metric by the `Invocations` metric to get an error rate. Then add the resulting time series to a graph on your CloudWatch dashboard.

You can also perform metric math programmatically, using the `GetMetricData` API operation. For more information, see GetMetricData.

Add a math expression to a CloudWatch graph

You can add a math expression to a graph on your CloudWatch dashboard. Each graph is limited to using a maximum of 500 metrics and expressions, so you can add a math expression only if the graph has 499 or fewer metrics. This applies even if not all the metrics are displayed on the graph.

To add a math expression to a graph

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Create or edit a graph. There needs to be at least one metric in the graph.
3. Choose **Graphed metrics**.
4. Choose **Math expression**, **Start with empty expression**. A new line appears for the expression.
5. In the new line, under the **Details** column, enter the math expression. The tables in the **Metric Math Syntax and Functions** section list the functions that you can use in the expression.

   To use a metric or the result of another expression as part of the formula for this expression, use the value shown in the **Id** column: for example, **m1+m2** or **e1-MIN(e1)**.

   You can change the value of **Id**. It can include numbers, letters, an underscore, and must start with a lowercase letter. Changing the value of **Id** to a more meaningful name can also make a graph easier to understand;for example, changing **m1** and **m2** to **errors** and **requests**.

   > **Tip**
   >
   > Choose the down arrow next to **Math Expression** to see a list of supported functions, which you can use when creating your expression.

6. For the **Label** column of the expression, enter a name that describes what the expression is calculating.

   If the result of an expression is an array of time series, each of those time series is displayed on the graph with a separate line, with different colors. Immediately under the graph is a legend for each line in the graph. For a single expression that produces multiple time series, the legend captions for those time series are in the format ***`Expression-Label Metric-Label`***. For example, if the graph includes a metric with a label of **Errors** and an expression

FILL(METRICS(), 0) that has a label of **Filled With 0:**, one line in the legend would be **Filled With 0: Errors**. To have the legend show only the original metric labels, set *`Expression-Label`* to be empty.

When one expression produces an array of time series on the graph, you can't change the colors used for each of those time series.

7. After you have added the desired expressions, you can simplify the graph by hiding some of the original metrics. To hide a metric or expression, clear the check box to the left of the **Id** field.

Metric math syntax and functions

The following sections explain the functions available for metric math. All functions must be written in uppercase letters (such as **AVG**), and the **Id** field for all metrics and math expressions must start with a lowercase letter.

The final result of any math expression must be a single time series or an array of time series. Some functions produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value in the time series.

Data type abbreviations

Some functions are valid for only certain types of data. The abbreviations in the following list are used in the tables of functions to represent the types of data supported for each function:

- **S** represents a scalar number, such as 2, -5, or 50.25.
- **TS** is a time series (a series of values for a single CloudWatch metric over time): for example, the `CPUUtilization` metric for instance `i-1234567890abcdef0` over the last three days.
- **TS[]** is an array of time series, such as the time series for multiple metrics.
- **String[]** is an array of strings.

The METRICS() function

The **METRICS()** function returns all the metrics in the request. Math expressions aren't included.

You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. For example, the expression **SUM(METRICS())** returns a time series (TS) that is the sum of the values of all the graphed metrics. **METRICS()/100** returns an array of time series, each of which is a time series showing each data point of one of the metrics divided by 100.

You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their **Id** field. You can also use **SUM([METRICS("4xx"), METRICS("5xx")])** to match multiple strings.

Basic arithmetic functions

The following table lists the basic arithmetic functions that are supported. Missing values in a time series are treated as 0. If the value of a data point causes a function to attempt to divide by zero, the data point is dropped.

Operation	Arguments	Examples
Arithmetic operators: + - * / ^	S, S	PERIOD(m1)/60
	S, TS	**5 * m1**
	TS, TS	**m1 - m2**
	S, TS[]	**SUM(100/[m1, m2])**
	TS, TS[]	**AVG(METRICS())**
		METRICS()*100
Unary subtraction -	S	**-5*m1**
	TS	**-m1**
	TS[]	**SUM(-[m1, m2])**

Comparison and logical operators

You can use comparison and logical operators with either a pair of time series or a pair of single scalar values. When you use a comparison operator with a pair of time series, the operators return

a time series where each data point is either 0 (false) or 1 (true). If you use a comparison operator on a pair of scalar values, a single scalar value is returned, either 0 or 1.

When comparison operators are used between two time series, and only one of the time series has a value for a particular time stamp, the function treats the missing value in the other time series as **0**.

You can use logical operators in conjunction with comparison operators, to create more complex functions.

The following table lists the operators that are supported.

Type of operator	Supported operators
Comparison operators	== != <= >= < >
Logical operators	AND or && OR or ||

To see how these operators are used, suppose you have two time series: **metric1** has values of `[30, 20, 0, 0]` and **metric2** has values of `[20, -, 20, -]` where `-` indicates that there is no value for that timestamp.

Expression	Output
(metric1 < metric2)	**0, 0, 1, 0**
(metric1 >= 30)	**1, 0, 0, 0**

Expression	Output
(metric1 > 15 AND metric2 > 15)	**1, 0, 0, 0**

Functions supported for metric math

The following table describes the functions that you can use in math expressions. Enter all functions in uppercase letters.

The final result of any math expression must be a single time series or an array of time series. Some functions in tables in the following sections produce a scalar number. You can use these functions within a larger function that ultimately produces a time series. For example, taking the **AVG** of a single time series produces a scalar number, so it can't be the final expression result. But you could use it in the function **m1-AVG(m1)** to display a time series of the difference between each individual data point and the average value of that data point.

In the following table, every example in the **Examples** column is an expression that results in a single time series or an array of time series. These examples show how functions that return scalar numbers can be used as part of a valid expression that produces a single time series.

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
ABS	TS TS[]	TS TS[]	Returns the absolute value of each data point.	**ABS(m1-m2)** **MIN(ABS([m1, m2]))** **ABS(METRICS())**	✓
ANOMALY_DETECTION_BAND	TS TS, S	TS[]	Returns an anomaly detection band for the specified metric. The band consists of two time series, one representing the upper	**ANOMALY_DETECTION_BAND(m1)** **ANOMALY_DETECTION_BAND(m1,4)**	

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			limit of the "normal" expected value of the metric, and the other representing the lower limit. The function can take two arguments . The first is the ID of the metric to create the band for. The second argument is the number of standard deviations to use for the band. If you don't specify this argument, the default of 2 is used. For more information, see Using CloudWatch anomaly detection.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
AVG	TS TS[]	S TS	The **AVG** of a single time series returns a scalar representing the average of all the data points in the metric. The **AVG** of an array of time series returns a single time series. Missing values are treated as 0. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `AVG(m2)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher	**SUM([m1,m2])/AVG(m2)** **AVG(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested. To use this function with alarms, especially alarms that have Auto Scaling actions, we recommend that you set the alarm to use M out of N datapoints, where M < N.		
CEIL	TS TS[]	TS TS[]	Returns the ceiling of each metric. The ceiling is the smallest integer greater than or equal to each value.	**CEIL(m1)** **CEIL(METRICS())** **SUM(CEIL(METRICS()))**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
DATAPOINT_COUNT	TS TS[]	S TS	Returns a count of the data points that reported values. This is useful for calculating averages of sparse metrics. **Note** We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when	**SUM(m1) / DATAPOINT_COUNT(m1)** **DATAPOINT_COUNT(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
DB_PERF_INSIGHTS	String, String, String String, String, String[]	TS (if given a single string) TS[] (if given an array of strings)	Returns Performance Insights Counter metrics for databases such as Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). This function returns the same amount of data that you can get by directly querying the Performance Insights APIs. You can use these metrics in CloudWatch for graphing and creating alarms. **Important** When you use this function, you must specify the Unique Database Resource ID of the database. This is different than	**DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', 'os.cpuUtilization.user.avg')** **DB_PERF_INSIGHTS('DOCDB', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', ['os.cpuUtilization.idle.avg', 'os.cpuUtilization.user.max'])**	

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			the database identifier. To find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is displayed in the **Configuration** section. **DB_PERF_INSIGHTS** also brings in the DBLoad metric at sub-minute intervals. Performance Insights metrics retrieved with this function are not stored in CloudWatch. Therefore, some CloudWatch features such as cross-acc		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			ount observability, anomaly detection, metric streams, metrics explorer, and Metric Insights don't work with Performance Insights metrics that you retrieve with **DB_PERF_INSIGHTS**. A single request using the **DB_PERF_INSIGHTS** function can retrieve the following numbers of data points. • 1080 data points for high-resolution periods (1s, 10s, 30s) • 1440 data points for standard-resolution periods (1m, 5m, 1hr, 1d) The **DB_PERF_INSIGHTS** function supports only the following period lengths: • 1 second		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			• 10 seconds • 30 seconds • 1 minute • 5 minutes • 1 hour • 1 day For more information about Amazon RDS Performance Insights counter metrics, see Performance Insights counter metrics. For more informati on about Amazon DocumentDB Performance Insights counter metrics, see Performance Insights for counter metrics. **Note** High-resolution metrics with sub-minute granularity retrieved by **DB_PERF_I**		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			NSIGHTS are only applicable to the **DBLoad** metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see Monitoring OS metrics with Enhanced Monitoring. You can create a high-resolution alarm using the **DB_PERF_INSIGHTS** function for a maximum time range of three hours. You can use the CloudWatc		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			h console to graph metrics retrieved with the **DB_PERF_INSIGHTS** function for any time range.		
DIFF	TS TS[]	TS TS[]	Returns the difference between each value in the time series and the preceding value from that time series.	**DIFF(m1)**	✓
DIFF_TIME	TS TS[]	TS TS[]	Returns the difference in seconds between the timestamp of each value in the time series and the timestamp of the preceding value from that time series.	**DIFF_TIME(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
FILL	TS, [S | REPEAT | LINEAR] TS[], [TS | S | REPEAT | LINEAR]	TS TS[]	Fills the missing values of a time series. There are several options for the values to use as the filler for missing values: • You can specify a value to use as the filler value. • You can specify a metric to use as the filler value. • You can use the **REPEAT** keyword to fill missing values with the most recent actual value of the metric before the missing value. • You can use the **LINEAR** keyword to fill the missing values with values that create a linear interpolation between the values at the beginning and the end of the gap.	**FILL(m1,10)** **FILL(METRICS(), 0)** **FILL(METRICS(), m1)** **FILL(m1, MIN(m1))** **FILL(m1, REPEAT)** **FILL(METRICS(), LINEAR)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			Note When you use this function in an alarm, you can encounter an issue if your metrics are being published with a slight delay, and the most recent minute never has data. In this case, **FILL** replaces that missing data point with the requested value. That causes the latest data point for the metric to always be the FILL value, which can result in the alarm being stuck in either OK state or ALARM state. You can work		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			around this by using a M out of N alarm. For more informati on, see Alarm evaluation.		
FIRST **LAST**	TS[]	TS	Returns the first or last time series from an array of time series. This is useful when used with the **SORT** function. It can also be used to get the high and low thresholds from the **ANOMALY_D ETECTION_BAND** function.	**IF(FIRST(SORT(METR ICS(), AVG, DESC))>10 0, 1, 0)** Looks at the top metric from an array, which is sorted by AVG. It then returns a 1 or 0 for each data point, depending on whether that data point value is more than 100. **LAST(ANOM ALY_DETEC TION_BAND(m1))** returns the upper bound of the anomaly prediction band.	✓
FLOOR	TS TS[]	TS TS[]	Returns the floor of each metric. The floor is the largest integer less than or equal to each value.	**FLOOR(m1)** **FLOOR(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
IF	**IF** expressi n	TS	Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series. For more information, see Using IF expressions.	For examples, see Using IF expressions.	✓
INSIGHT_RULE_METRIC	**INSIGHT ULE_MI C(ruleN e, metricN e)**	TS	Use **INSIGHT_RULE_METRIC** to extract statistics from a rule in Contributor Insights. For more information, see Graphing metrics generated by rules in CloudWatch.		
LAMBDA	**LAMBD mbdaFu ionNam [, optiona arg]*)**	TS TS[]	Calls a Lambda function to query metrics from a data source that is not CloudWatch. For more information, see How to pass arguments to your Lambda function.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
LOG	TS TS[]	TS TS[]	The **LOG** of a time series returns the natural logarithm value of each value in the time series.	**LOG(METRICS())**	✓
LOG10	TS TS[]	TS TS[]	The **LOG10** of a time series returns the base-10 logarithm value of each value in the time series.	**LOG10(m1)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
MAX	TS TS[]	S TS	The **MAX** of a single time series returns a scalar representing the maximum value of all data points in the metric. If you input an array of time series, the **MAX** function creates and returns a time series that consists of the highest value for each data point, among the time series that were used as the input. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `MAX(m2)` Whenever an	**MAX(m1)/m1** **MAX(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		
METRIC_COUNT	TS[]	S	Returns the number of metrics in the time series array.	**m1/METRIC_COUNT(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
METRICS	null string	TS[]	The **METRICS()** function returns all CloudWatch metrics in the request. Math expressions aren't included. You can use **METRICS()** within a larger expression that produces a single time series or an array of time series. You can use the **METRICS()** function with a string to return only the graphed metrics that contain that string in their **Id** field. For example, the expression **SUM(METRICS("errors"))** returns a time series that is the sum of the values of all the graphed metrics that have 'errors' in their **Id** field. You can also use **SUM([METRICS("4xx"), METRICS("5xx")])** to match multiple strings.	**AVG(METRICS())** **SUM(METRICS("errors"))**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
MIN	TS TS[]	S TS	The **MIN** of a single time series returns a scalar representing the minimum value of all data points in the metric. If you input an array of time series, the **MIN** function creates and returns a time series that consists of the lowest value for each data point, among the time series that were used as the input. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `MIN(m2)` Whenever an	**m1-MIN(m1)** **MIN(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
MINUTE **HOUR** **DAY** **DATE** **MONTH** **YEAR** **EPOCH**	TS	TS	These functions take the period and range of the time series and return a new non-sparse time series where each value is based on its timestamp. • **MINUTE** returns a non-sparse time series of integers between 0 and 59 that represent the UTC minute of each timestamp in the original time series. • **HOUR** returns a non-sparse time series of integers between 0 and 23 that represent the UTC hour of each timestamp in the original time series. • **DAY** returns a non-sparse time series of integers between 1 and 7 that represent the UTC day of the week of each timestamp in the	**MINUTE(m1)** **IF(DAY(m1)<6,m1)** returns metrics only from weekdays, Monday to Friday. **IF(MONTH(m1) == 4,m1)** returns only metrics published in April.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			original time series. 1 represents Monday and 7 represents Sunday. • **DATE** returns a non-sparse time series of integers between 1 and 31 that represent the UTC day of the month of each timestamp in the original time series. • **MONTH** returns a non-sparse time series of integers between 1 and 12 that represent the UTC month of each timestamp in the original time series. 1 represents January and 12 represents December. • **YEAR** returns a non-sparse time series of integers that represent the UTC year of each timestamp in the original time series.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			• **EPOCH** returns a non-sparse time series of integers that represent the UTC time in seconds since the Epoch of each timestamp in the original time series. The Epoch is January 1, 1970.		
PERIOD	TS	S	Returns the period of the metric in seconds. Valid input is metrics, not the results of other expressions.	**m1/PERIOD(m1)**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
RATE	TS TS[]	TS TS[]	Returns the rate of change of the metric per second. This is calculated as the difference between the latest data point value and the previous data point value, divided by the time difference in seconds between the two values. **Important** Setting alarms on expressions that use the RATE function on metrics with sparse data can behave unpredictably, because the range of data points fetched when evaluating the alarm can vary based on when the data	**RATE(m1)** **RATE(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			points were last published.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
REMOVE_EMPTY	TS[]	TS[]	Removes any time series that have no data points from an array of time series. The result is an array of time series where each time series contains at least one data point. **Note** We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts	**REMOVE_EMPTY(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
RUNNING_SUM	TS TS[]	TS TS[]	Returns a time series with the running sum of the values in the original time series. **Note** We recommend that you do not use this function in CloudWatch alarms. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.	**RUNNING_SUM([m1,m2])**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
SEARCH	Search expressi n	One or more TS	Returns one or more time series that match a search criteria that you specify. The **SEARCH** function enables you to add multiple related time series to a graph with one expression. The graph is dynamically updated to include new metrics that are added later and match the search criteria. For more information, see Use search expressions in graphs. You can't create an alarm based on a **SEARCH** expression. This is because search expressions return multiple time series, and an alarm based on a math expression can watch only one time series. If you are signed in to a monitoring account in CloudWatch cross-		✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			account observability, the **SEARCH** function finds metrics in the source accounts and the monitoring account.		
SERVICE_QUOTA	TS that is a usage metric	TS	Returns the service quota for the given usage metric. You can use this to visualize how your current usage compares to the quota, and to set alarms that warn you when you approach the quota. For more information, see AWS usage metrics.		✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
SLICE	(TS[], S, S) or (TS[], S)	TS[] TS	Retrieves part of an array of time series. This is especially useful when combined with **SORT**. For example, you can exclude the top result from an array of time series. You can use two scalar arguments to define the set of time series that you want returned. The two scalars define the start (inclusive) and end (exclusive) of the array to return. The array is zero-indexed, so the first time series in the array is time series 0. Alternatively, you can specify just one value, and CloudWatch returns all time series starting with that value.	**SLICE(SORT(METRICS(), SUM, DESC), 0, 10)** returns the 10 metrics from the array of metrics in the request that have the highest SUM value. **SLICE(SORT(METRICS(), AVG, ASC), 5)** sorts the array of metrics by the AVG statistic, then returns all the time series except for the 5 with the lowest AVG.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
SORT	(TS[], FUNCTI SORT_C R) (TS[], FUNCTI SORT_C R, S)	TS[]	Sorts an array of time series according to the function you specify. The function you use can be **AVG**, **MIN**, **MAX**, or **SUM**. The sort order can be either **ASC** for ascending (lowest values first) or **DESC** to sort the higher values first. You can optionall y specify a number after the sort order which acts as a limit. For example, specifying a limit of 5 returns only the top 5 time series from the sort. When this math function is displayed on a graph, the labels for each metric in the graph are also sorted and numbered.	**SORT(METRICS(), AVG, DESC, 10)** calculates the average value of each time series, sorts the time series with the highest values at the beginning of the sort, and returns only the 10 time series with the highest averages. **SORT(METRICS(), MAX, ASC)** sorts the array of metrics by the MAX statistic, then returns all of them in ascending order.	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
STDDEV	TS TS[]	S TS	The **STDDEV** of a single time series returns a scalar representing the standard deviation of all data points in the metric. The **STDDEV** of an array of time series returns a single time series. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `STDDEV(m2)` Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data	**m1/STDDEV(m1)** **STDDEV(METRICS())**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
SUM	TS TS[]	S TS	The **SUM** of a single time series returns a scalar representing the sum of the values of all data points in the metric. The **SUM** of an array of time series returns a single time series. **Note** We recommend that you do not use this function in CloudWatch alarms if you want the function to return a scalar. For example, `SUM(m1)`. Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data	**SUM(METRICS())/SUM(m1)** **SUM([m1,m2])** **SUM(METRICS("errors"))/SUM(METRICS("requests"))*100**	✓

Function	Argume	Return type*	Description	Examples	Supported for cross-account?
			points than the number specified as Evaluation Periods. This function acts differently when extra data is requested.		
TIME_SERIES	S	TS	Returns a non-sparse time series where every value is set to a scalar argument.	**TIME_SERIES(MAX(m1))** **TIME_SERIES(5*AVG(m1))** **TIME_SERIES(10)**	✓

*Using a function that returns only a scalar number is not valid, as all final results of expressions must be a single time series or an array of time series. Instead, use these functions as part of a larger expression that returns a time series.

Using IF expressions

Use **IF** along with a comparison operator to filter out data points from a time series, or create a mixed time-series composed of multiple collated time series.

IF uses the following arguments:

```
IF(condition, trueValue, falseValue)
```

The condition evaluates to FALSE if the value of the condition data point is 0, and to TRUE if the value of the condition is any other value, whether that value is positive or negative. If the condition is a time series, it is evaluated separately for every timestamp.

The following lists the valid syntax's. For each of these syntax's, the output is a single time series.

- **IF(TS *Comparison Operator* S, S | TS, *S* / *TS*)**

  > **Note**
  >
  > If the `TS comparison operator S` is TRUE but `metric2` doesn't have a corresponding data point, the output will be 0.

- **IF(TS, TS, *TS*)**
- **IF(TS, S, *TS*)**
- **IF(TS, TS, S)**
- **IF(TS, S, S)**
- **IF(S, TS, *TS*)**

The following sections provide more details and examples for these syntax's.

IF(TS *Comparison Operator* S, scalar2 | metric2, scalar3 | metric3)

The corresponding output time series value:

- has the value of **scalar2** or **metric2**, if **TS *Comparison Operator* S** is TRUE
- has the value of **scalar3** or **metric3**, if **TS *Comparison Operator* S** is FALSE
- has the value of **0** if the **TS *Comparison Operator*** is TRUE and the corresponding data point in **metric2** doesn't exist.
- has the value of **0** if the **TS *Comparison Operator*** is FALSE and the corresponding data point in **metric3** doesn't exist.
- is an empty time series, if the corresponding data point of does not exist in **metric3**, or if **scalar3**/**metric3** is omitted from the expression

IF(metric1, metric2, *metric3*)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- has the value of **0**, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist in **metric3**
- is dropped, if the corresponding data point of **metric1** is FALSE and **metric3** is omitted from the expression.
- is dropped, if the corresponding data point of **metric1** is missing.

The following table shows an example for this syntax.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**
(metric2)	**[30, -, 0, 0, 30]**
(metric3)	**[0, 0, 20, -, 20]**
IF(metric1, metric2, metric3)	**[30, 0, 20, 0, -]**

IF(metric1, scalar2, metric3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **scalar2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **metric3**, if the corresponding data point of **metric1** is FALSE.
- is dropped, if the corresponding data point of **metric1** is FALSE and the corresponding data point does not exist on **metric3**, or if **metric3** is omitted from the expression.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**

Metric or function	Values
scalar2	**5**
(metric3)	**[0, 0, 20, -, 20]**
IF(metric1, scalar2, metric3)	**[5, 5, 20, -, -]**

IF(metric1, metric2, scalar3)

For each data point of **metric1**, the corresponding output time series value:

- has the value of **metric2**, if the corresponding data point of **metric1** is TRUE.
- has the value of **scalar3**, if the corresponding data point of **metric1** is FALSE.
- has the value of **0**, if the corresponding data point of **metric1** is TRUE and the corresponding data point does not exist in **metric2**.
- is dropped if the corresponding data point in **metric1** does not exist.

Metric or function	Values
(metric1)	**[1, 1, 0, 0, -]**
(metric2)	**[30, -, 0, 0, 30]**
scalar3	**5**
IF(metric1, metric2, scalar3)	**[30, 0, 5, 5, -]**

IF(scalar1, metric2, *metric3*)

The corresponding output time series value:

- has the value of **metric2**, if **scalar1** is TRUE.
- has the value of **metric3**, if **scalar1** is FALSE.
- is an empty time series, if **metric3** is omitted from the expression.

Use case examples for IF expressions

The following examples illustrate the possible uses of the **IF** function.

- To display only the low values of a metric:

  IF(metric1<400, metric1)
- To change each data point in a metric to one of two values, to show relative highs and lows of the original metric:

  IF(metric1<400, 10, 2)
- To display a 1 for each timestamp where latency is over the threshold, and display a 0 for all other data points:

  IF(latency>threshold, 1, 0)

Use metric math with the GetMetricData API operation

You can use `GetMetricData` to perform calculations using math expressions, and also retrieve large batches of metric data in one API call. For more information, see GetMetricData.

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions.

Anomaly detection on metric math doesn't support the following functions:

- Expressions that contain more than one **ANOMALY_DETECTION_BAND** in the same line.
- Expressions that contain more than 10 metrics or math expressions.
- Expressions that contain the **METRICS** expression.
- Expressions that contain the **SEARCH** function.
- Expressions that use the **DB_PERF_INSIGHTS** function.
- Expressions that use metrics with different periods.
- Expressions that use periods longer than one hour as input.
- Expressions that use high-resolution metrics as input.

For more information about this feature, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

Using CloudWatch anomaly detection

When you enable *anomaly detection* for a metric, CloudWatch applies statistical and machine learning algorithms. These algorithms continuously analyze metrics of systems and applications, determine normal baselines, and surface anomalies with minimal user intervention.

The algorithms generate an anomaly detection model. The model generates a range of expected values that represent normal metric behavior.

You can enable anomaly detection using the AWS Management Console, the AWS CLI, CloudFormation, or the AWS SDK. You can enable anomaly detection on metrics vended by AWS and also on custom metrics. In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

You can use the model of expected values in two ways:

- Create anomaly detection alarms based on a metric's expected value. These types of alarms don't have a static threshold for determining alarm state. Instead, they compare the metric's value to the expected value based on the anomaly detection model.

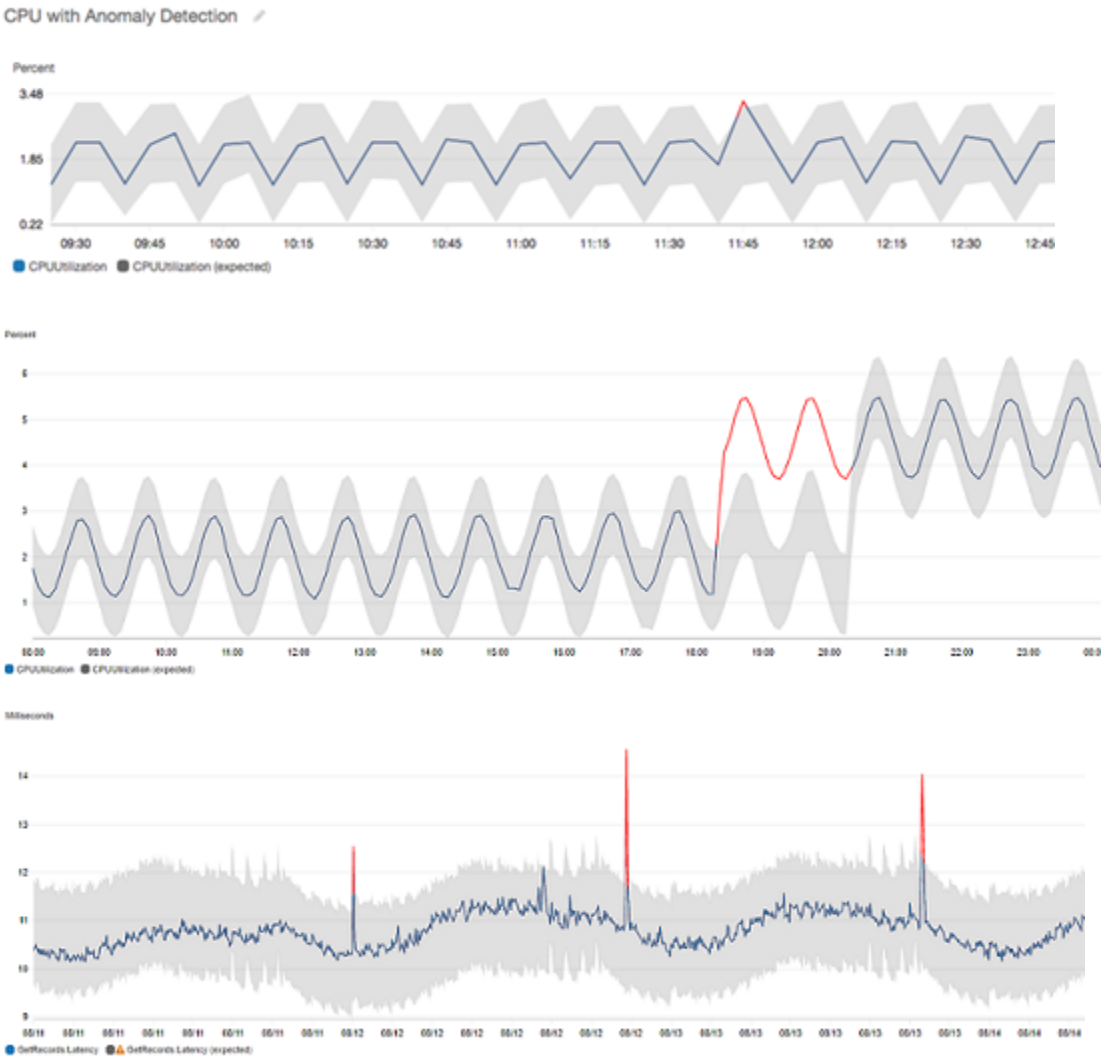
  You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or both.

  For more information, see Create a CloudWatch alarm based on anomaly detection.
- When viewing a graph of metric data, overlay the expected values onto the graph as a band. This makes it visually clear which values in the graph are out of the normal range. For more information, see Creating a graph.

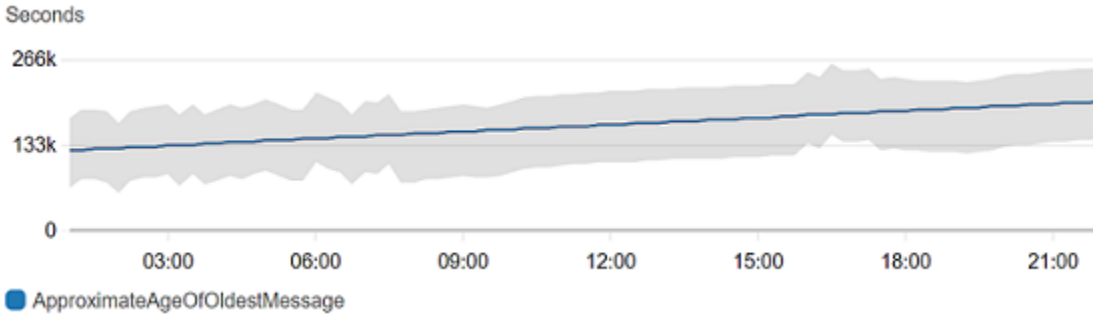
  You can also retrieve the upper and lower values of the model's band by using the `GetMetricData` API request with the `ANOMALY_DETECTION_BAND` metric math function. For more information, see GetMetricData.

In a graph with anomaly detection, the expected range of values is shown as a gray band. If the metric's actual value goes beyond this band, it is shown as red during that time.

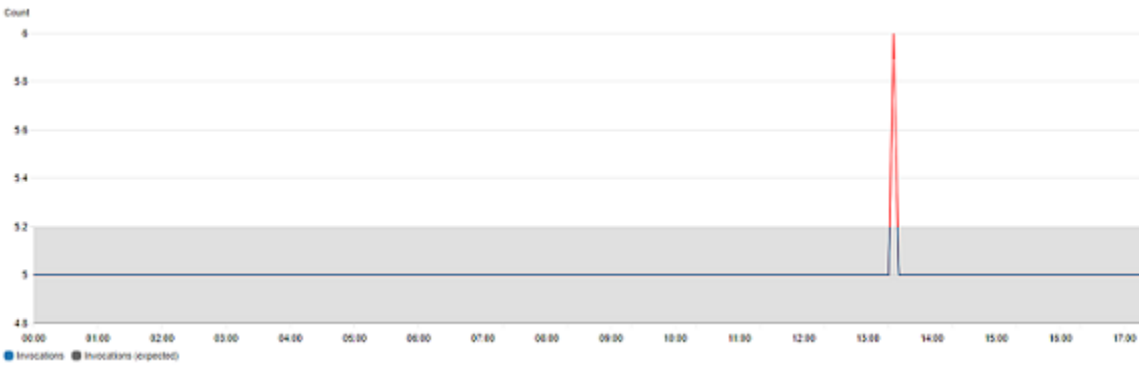
Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes could be hourly, daily, or weekly, as shown in the following examples.



The longer-range trends could be downward or upward.



Anomaly detection also works well with metrics with flat patterns.



How CloudWatch anomaly detection works

When you enable anomaly detection for a metric, CloudWatch applies machine learning algorithms to the metric's past data to create a model of the metric's expected values. The model assesses both trends and hourly, daily, and weekly patterns of the metric. The algorithm trains on up to two weeks of metric data, but you can enable anomaly detection on a metric even if the metric does not have a full two weeks of data.

You specify a value for the anomaly detection threshold that CloudWatch uses along with the model to determine the "normal" range of values for the metric. A higher value for the anomaly detection threshold produces a thicker band of "normal" values.

The machine learning model is specific to a metric and a statistic. For example, if you enable anomaly detection for a metric using the `AVG` statistic, the model is specific to the `AVG` statistic.

When CloudWatch creates a model for many common metrics from AWS services, it ensures that the band doesn't extend outside of logical values. For example, the band for `MemoryUtilization` of an EC2 instance will stay between 0 and 100, and the bands tracking CloudFront `Requests`, which can't be negative, will never extend below zero.

After you create a model, CloudWatch anomaly detection continually evaluates the model and makes adjustments to it to make sure that it is as accurate as possible. This includes re-training the model to adjust if the metric values evolve over time or have sudden changes, and also includes predictors to improve the models of metrics that are seasonal, spiky, or sparse.

After you enable anomaly detection on a metric, you can choose to exclude specified time periods of the metric from being used to train the model. This way, you can exclude deployments or other unusual events from being used for model training, ensuring the most accurate model is created.

Using anomaly detection models for alarms incurs charges on your AWS account. For more information, see Amazon CloudWatch Pricing.

Anomaly detection on metric math

Anomaly detection on metric math is a feature that you can use to create anomaly detection alarms on the output metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands. The feature supports basic arithmetic functions, comparison and logical operators, and most other functions. For information about functions that are not supported, see Using metric math in the *Amazon CloudWatch User Guide*.

You can create anomaly detection models based on metric math expressions similar to how you already create anomaly detection models. From the CloudWatch console, you can apply anomaly detection to metric math expressions and select anomaly detection as a threshold type for these expressions.

> **Note**
>
> Anomaly detection on metric math only can be enabled and edited in the latest version of the metrics user interface. When you create anomaly detectors based on metric math expressions in the new version of the interface, you can view them in the old version, but not edit them.

For information about how to create, edit, and delete alarms and models for anomaly detection and metric math, see the following sections:

- Create a CloudWatch alarm based on anomaly detection
- Editing an anomaly detection model
- Deleting an anomaly detection model
- Improving anomaly detection alarm accuracy with feedback
- Creating a CloudWatch alarm based on a metric math expression

You also can create, delete, and discover anomaly detection models based on metric math expressions using the CloudWatch API with `PutAnomalyDetector`, `DeleteAnomalyDetector`, and `DescribeAnomalyDetectors`. For information about these API actions, see the following sections in the *Amazon CloudWatch API Reference*.

- PutAnomalyDetector
- DeleteAnomalyDetector

- DescribeAnomalyDetectors

For information about how anomaly detection alarms are priced, see Amazon CloudWatch pricing.

Query metrics from other data sources

You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources. To do so, you connect CloudWatch to the other data sources. This gives you a single, consolidated monitoring experience within the CloudWatch console. You can have a unified view of your infrastructure and application metrics regardless of where the data is stored, helping you identify and resolve issues faster.

After you connect to a data source using a CloudWatch wizard, CloudWatch creates an AWS CloudFormation stack that deploys and configures an AWS Lambda function. This Lambda function runs on demand every time you query the data source. The CloudWatch query builder shows you in real time a list of elements that can be queried, such as metrics, tables, fields, or labels. As you make choices, the query builder pre-populates a query in the native language of the selected source.

CloudWatch provides guided wizards for you to connect to the following data sources. For these data sources, you provide basic information to identify the data source and credentials. You can also manually create connectors to other data sources by creating your own Lambda functions.

- Amazon OpenSearch Service– Derive metrics from your OpenSearch Service logs and traces.
- Amazon Managed Service for Prometheus– Query these metrics using PromQL.
- Amazon RDS for MySQL – Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon RDS for PostgreSQL– Use SQL to convert data stored in your Amazon RDS tables into metrics.
- Amazon S3 CSV files– Display metrics data from a CSV file stored in an Amazon S3 bucket.
- Microsoft Azure Monitor– Query metrics from your Microsoft Azure Monitor account.
- Prometheus– Query these metrics using PromQL.

After you create connectors to data sources, see Creating a graph of metrics from another data source for information about graphing a metric from a data source. For information about setting an alarm on a metric from a data source, see Create an alarm based on a connected data source.

Topics

- Managing access to data sources
- Connect to a prebuilt data source with a wizard
- Create a custom connector to a data source
- Use your custom data source
- Delete a connector to a data source

Managing access to data sources

CloudWatch uses CloudFormation to create the required resources in your account. We recommend that you use the `cloudformation:TemplateUrl` condition to control access to CloudFormation templates when you grant `CreateStack` permissions to IAM users.

> **Warning**
>
> Any user that you grant data source invoke permission to can query metrics from that data source even if that user does not have direct IAM permissions to the data source. For example, if you grant `lambda:InvokeFunction` permissions on a Amazon Managed Service for Prometheus data source Lambda function to a user, that user will be able to query metrics from the corresponding Amazon Managed Service for Prometheus workspace even if you didn't grant them direct IAM access to that workspace.

You can find template URLs for data sources on the **Create stack** page in the CloudWatch Settings Console.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFormationCreateStack",
            "Effect": "Allow",
            "Action": [
```

```
                "cloudformation:CreateStack"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudformation:TemplateUrl": [
                        "https://s3.us-east-1.amazonaws.com/amzn-s3-demo-bucket/
template.json"
                    ]
                }
            }
        }
    ]
}
```

For more information about controlling CloudFormation access, see Controlling access with AWS Identity and Access Management

Connect to a prebuilt data source with a wizard

This topic provides instructions for using the wizard to connect CloudWatch to the following data sources.

- Amazon OpenSearch Service
- Amazon Managed Service for Prometheus
- Amazon RDS for MySQL
- Amazon RDS for PostgreSQL
- Amazon S3 CSV files
- Microsoft Azure Monitor
- Prometheus

The subsections in this topic include notes about managing and querying with each of these data sources.

To create a connector to a data source

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Select the source that you want, then choose **Next**.
6. Enter a name for the data source.
7. Enter the other required information, depending on the data source that you chose. This can include credentials to access the data source and data source identifying information such as Prometheus workspace name, database name, or Amazon S3 bucket name. For AWS services, the wizard discovers the resources and populates them into the selection dropdown.

   For more notes about the data source you are using, see the sections after this procedure.
8. To have CloudWatch connect to the data source in a VPC, choose **Use a VPC** and select the VPC to use. Then select the subnet and security group.
9. Choose **I acknowledge that CloudFormation will create IAM resources**. This resource is the Lambda function execution role.
10. Choose **Create data source**.

    The new source that you just added doesn't appear until the CloudFormation stack is done creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

    When your new data source appears in this list, it is ready to use. You can choose **Query from CloudWatch metrics** to begin querying with it. For more information, see Creating a graph of metrics from another data source.

Amazon Managed Service for Prometheus

Updating the data source configuration

- You can update your data source manually by doing the following:
  - To update the Amazon Managed Service for Prometheus workspace ID, update the `AMAZON_PROMETHEUS_WORKSPACE_ID` environment variable for the data source connector Lambda function.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying Amazon Managed Service for Prometheus, after you select the data source in the **Multi source query** tab and select an Amazon Managed Service for Prometheus connector, you can use the **Query helper** to discover metrics and labels and provide simple PromQL queries. You can also use the PromQL query editor to build a PromQL query.
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Amazon OpenSearch Service

Creating the data source

If the OpenSearch domain is enabled for FGAC, you must map the execution role of the connector Lambda function to a user in OpenSearch Service. For more information, see the **Mapping users to roles** section in Managing permissions in the OpenSearch Service documentation.

If your OpenSearch domain is only accessible within a Virtual Private Cloud (VPC), you need to manually include a new environment variable in the Lambda function called `AMAZON_OPENSEARCH_ENDPOINT`. The value for this variable should be the root domain of the OpenSearch endpoint. You can obtain this root domain by removing `https://` and `<region>.es.amazonaws.com` from the domain endpoint listed in the OpenSearch Service console. For example, if your domain endpoint is `https://sample-domain.us-east-1.es.amazonaws.com`, the root domain would be `sample-domain`.

Updating the data source

- You can update your data source manually by doing the following:
  - To update the OpenSearch Service domain, update the `AMAZON_OPENSEARCH_DOMAIN_NAME` environment variable for the data source connector Lambda function.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying OpenSearch Service, after you select the data source in the **Multi source query** tab, do the following:

- Select the Index to query.
- Select the Metric name (Any numeric field in the document) and Stat.
- Select the Time axis (Any date field in the document).
- Select Filters to apply (Any String field in the document).
- Choose **Graph query**.

Amazon RDS for PostgreSQL and Amazon RDS for MySQL

Creating the data source

- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in Connect to a prebuilt data source with a wizard. If the data source is to connect to the VPC for credentials, the endpoint must be configured in the VPC. For more information, see Using an AWS Secrets Manager VPC endpoint.

  Additionally, you must create a VPC endpoint for the Amazon RDS service. For more information, see Amazon RDS API and interface VPC endpoints (AWS PrivateLink).

Updating the data source

- You can update your data source manually by doing the following:
  - To update the database instance, update the `RDS_INSTANCE` environment variable for the data source connector Lambda function.
  - To update the username and password used to connect to Amazon RDS, use AWS Secrets Manager. You can find the ARN of the secret used for the data source in the environment variable `RDS_SECRET` on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see Modify an AWS Secrets Manager secret.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

- When querying Amazon RDS, after you select the data source in the **Multi source query** tab and select an Amazon RDS connector, you can use the database discoverer to view available databases, tables, and columns. You can also use the SQL editor to create an SQL query.

  You can use the following variables in the query:

- `$start.iso` – The start time in ISO date format
- `$end.iso` – The end time in ISO date format
- `$period` – The selected period in seconds

  For example, you can query `SELECT value, timestamp FROM table WHERE timestamp BETWEEN $start.iso and $end.iso`
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

> **Note**
>
> If no date field is found in the results, the values for each numeric field are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using `SUM` and aligned with the period in CloudWatch.

Amazon S3 CSV files

Querying the data source

- When querying Amazon S3 CSV files, after you select the data source in the **Multi source query** tab and select an Amazon S3 connector, you select the Amazon S3 bucket and key.

  The CSV file must be formatted in the following ways:
  - The time stamp must be the first column.
  - The table must have a header row. The headers are used to name your metrics. The title of the time stamp column will be ignored, only the titles of the metrics columns are used.
  - The time stamps must be in ISO date format.
  - The metrics must be numeric fields.

```
Timestamp, Metric-1, Metric-2, ...
```

The following is an example:

timestamp	CPU (%)	Memory (%)	Storage (%)
2023-11-23T17:09:41+00:00	1	2	3
2023-11-23T17:04:41+00:00	4	5	6
2023-11-23T16:59:41+00:00	7	8	9
2023-11-23T16:54:41+00:00	10	11	12

Note

If no timestamp is provided, the values for each metric are summed to single values and plotted across the provided time range. If the timestamps don't align with the selected period in CloudWatch, the data is automatically aggregated using `SUM` and aligned with the period in CloudWatch.

Microsoft Azure Monitor

Creating the data source

- You must provide your tenant ID, client ID, and client secret to connect to Microsoft Azure Monitor. The credentials will be stored in AWS Secrets Manager. For more information, see Create a Microsoft Entra application and service principal that can access resources in the Microsoft documentation.

Updating the data source

- You can update your data source manually by doing the following:

- To update the tenant ID, client ID, and client secret used to connect to Azure Monitor, you can find the ARN of the secret used for the data source as the `AZURE_CLIENT_SECRET` environment variable on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see Modify an AWS Secrets Manager secret.

Querying the data source

- When querying Azure Monitor, after you select the data source in the **Multi source query** tab and select an Azure Monitor connector, you specify the Azure subscription, and the resource group and resource. You can then select the metric namespace, metric, and aggregation, and filter by dimensions.

Prometheus

Creating the data source

- You must provide the Prometheus endpoint and the user and password required to query Prometheus. The credentials will be stored in AWS Secrets Manager.
- If your data source is only accessible in a VPC, you must include the VPC configuration for the connector, as described in Connect to a prebuilt data source with a wizard. If the data source is to connect to for credentials, the endpoint must be configured in the VPC. For more information, see Using an AWS Secrets Manager VPC endpoint.

Updating data source configuration

- You can update your data source manually by doing the following:
  - To update the Prometheus endpoint, specify the new endpoint as the `PROMETHEUS_API_ENDPOINT` environment variable on the data source Lambda function.
  - To update the username and password used to connect to Prometheus, you can find the ARN of the secret used for the data source as the `PROMETHEUS_API_SECRET` environment variable on the data source Lambda function. For more information about updating the secret in AWS Secrets Manager, see Modify an AWS Secrets Manager secret.
  - To update the VPC configuration, see Configuring VPC access (console) for more information.

Querying the data source

Important

Prometheus metric types are different than CloudWatch metrics and many metrics available through Prometheus are cumulative by design. When you query Prometheus metrics, CloudWatch doesn't apply any additional transformation to the data: if you specify only the metric name or label, the displayed value will be cumulative. For more information, see Metric types in the Prometheus documentation.
To see Prometheus metrics data as discrete values, like CloudWatch metrics, you need to edit the query before you run it. For example, you might need to add a call to the rate function over your Prometheus metric name. For documentation about the rate function and other Prometheus functions, see rate() in the Prometheus documentation.

Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid. For example, if your query contains a single line comment it will not be valid. If you try to create a dashboard or alarm with a multi-line query from the command line or Infrastructure as Code, the API will reject the action with a parse error.

Notification of Available Updates

From time to time, Amazon might notify you that we recommend that you update your connectors with a newer available version and will provide instructions for how to do so.

Create a custom connector to a data source

This topic describes how to connect a custom data source to CloudWatch. You can connect a custom data source to CloudWatch in two ways:

- Using a sample template that CloudWatch provides. You can use either JavaScript or Python with this template. These templates include sample Lambda code that will be useful to you when you create your Lambda function. You can then modify the Lambda function from the template to connect to your custom data source.
- Creating an AWS Lambda function from scratch that implements the data source connector, the data query, and the preparation of the time series for use by CloudWatch. This function must pre-aggregate or merge datapoints if needed, and also align the period and timestamps to be compatible with CloudWatch.

Contents

- Use a template
- Create a custom data source from scratch
  - Step 1: Create the function
    - GetMetricData event
    - DescribeGetMetricData event
    - Important considerations for CloudWatch alarms
    - (Optional) Use AWS Secrets Manager to store credentials
    - (Optional) Connect to a data source in a VPC
  - Step 2: Create a Lambda permissions policy
  - Step 3: Attach a resource tag to the Lambda function

Use a template

Using a template creates a sample Lambda function, and can help you get your custom connector built faster. These sample functions provide sample code for many common scenarios involved with building a custom connector. You can examine the Lambda code after you create a connector with a template, then modify it to use to connect to your data source.

Additionally, if you use the template, CloudWatch takes care of creating the Lambda permissions policy and attaching resource tags to the Lambda function.

To use the template to create a connector to a custom data source

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Create data source**.
5. Choose the radio button for **Custom - getting started template** and then choose **Next**.
6. Enter a name for the data source.
7. Select the one of the listed templates.
8. Select either Node.js or Python.
9. Choose **Create data source**.

The new custom source that you just added doesn't appear until the CloudFormation stack finishes creating it. To check progress, you can choose **View the status of my CloudFormation stack**. Or you can choose the refresh icon to update this list.

When your new data source appears in this list, it is ready for you to test in the console and modify.

10. (Optional) To query the test data from this source in the console, follow the instructions in Creating a graph of metrics from another data source.
11. Modify the Lambda function for your needs.

    a. In the navigation pane, choose **Settings**.

    b. Choose the **Metrics data sources** tab.

    c. Choose **View in Lambda console** for the source that you want to modify.

    You can now modify the function to access your data source. For more information, see Step 1: Create the function.

    > **Note**
    >
    > By using the template, when you write your Lambda function you don't need to follow the instructions in Step 2: Create a Lambda permissions policy or Step 3: Attach a resource tag to the Lambda function. These steps were performed by CloudWatch because you used the template.

Create a custom data source from scratch

Follow the steps in this section to create a Lambda function that connects CloudWatch to a data source.

Step 1: Create the function

A custom data source connector must support `GetMetricData` events from CloudWatch. Optionally, you can also implement a `DescribeGetMetricData` event to provide documentation to users in the CloudWatch console for how to use the connector. The `DescribeGetMetricData` response can also be used to set defaults that are used in the CloudWatch custom query builder.

CloudWatch provides code snippets as samples to help you get started. For more information, see the samples repository at https://github.com/aws-samples/cloudwatch-data-source-samples.

Constraints

- The response from Lambda must be smaller than 6 Mb. If the response exceeds 6 Mb, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must complete its execution within 10 seconds for visualization and dashboarding purposes, or within 4.5 seconds for alarms usage. If the execution time exceeds that time, the `GetMetricData` response marks the Lambda function as `InternalError` and no data is returned.
- The Lambda function must send its output using epoch timestamps in seconds.
- If the Lambda function doesn't resample the data and instead returns data that doesn't correspond to the start time and period length that was requested by the CloudWatch user, that data is ignored by CloudWatch. The extra data is discarded from any visualization or alarming. Any data that is not between the start time and end time is also discarded.

  For example, if a user asks for data from 10:00 to 11:00 with a period of 5 min, then "10:00:00 to 10:04:59" and "10:05:00 to 10:09:59" are the valid time ranges for data to be returned. You must return a time series that includes `10:00 value1`, `10:05 value2`, and so on. If the function returns `10:03 valueX`, for example, it gets dropped because 10:03 does not correspond to the requested start time and period.
- Multi-line queries are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

GetMetricData event

Request payload

The following is an example of a `GetMetricData` request payload sent as input to the Lambda function.

```
{
  "EventType": "GetMetricData",
  "GetMetricDataRequest": {
    "StartTime": 1697060700,
```

```
    "EndTime": 1697061600,
    "Period": 300,
    "Arguments": ["serviceregistry_external_http_requests{host_cluster!=\"prod\"}"]
  }
}
```

- **StartTime**– The timestamp specifying the earliest data to return. The **Type** is timestamp epoch seconds.
- **EndTime**– The timestamp specifying the latest data to return. The **Type** is timestamp epoch seconds.
- **Period**– The number of seconds that each aggregation of the metrics data represents. The minimum is 60 seconds. The **Type** is Seconds.
- **Arguments**– An array of arguments to pass to the Lambda metric math expression. For more information about passing arguments, see How to pass arguments to your Lambda function.

Response payload

The following is an example of a `GetMetricData` response payload returned by the Lambda function.

```
{
   "MetricDataResults": [
      {
         "StatusCode": "Complete",
         "Label": "CPUUtilization",
         "Timestamps": [ 1697060700, 1697061000, 1697061300 ],
         "Values": [ 15000, 14000, 16000 ]
      }
   ]
}
```

The response payload will contain either a `MetricDataResults` field or an `Error` field, but not both.

A `MetricDataResults` field is a list of time-series fields of type `MetricDataResult`. Each of those time-series fields can include the following fields.

- **StatusCode**– (Optional) `Complete` indicates that all data points in the requested time range were returned. `PartialData` means that an incomplete set of data points were returned. If this is omitted, the default is `Complete`.

  Valid Values: `Complete` | `InternalError` | `PartialData` | `Forbidden`
- **Messages**– Optional list of messages with additional information about the data returned.

  Type: Array of MessageData objects with `Code` and `Value` strings.
- **Label**– The human-readable label associated with the data.

  Type: String
- **Timestamps**– The timestamps for the data points, formatted in epoch time. The number of timestamps always matches the number of values and the value for `Timestamps[x]` is `Values[x]`.

  Type: Array of timestamps
- **Values**– The data point values for the metric, corresponding to `Timestamps`. The number of values always matches the number of timestamps and the value for `Timestamps[x]` is `Values[x]`.

  Type: Array of doubles

For more information about `Error` objects, see the following sections.

Error response formats

You can optionally use the error response to provide more information about errors. We recommend that you return an error with Code Validation when a validation error occurs, such as when a parameter is missing or is the wrong type.

The following is an example of the response when the Lambda function wants to raise a `GetMetricData` validation exception.

```
{
   "Error": {
      "Code": "Validation",
      "Value": "Invalid Prometheus cluster"
   }
}
```

The following is an example of the response when the Lambda function indicates that it's unable to return data because of an access issue. The response is translated into a single time series with a status code of `Forbidden`.

```
{
   "Error": {
      "Code": "Forbidden",
      "Value": "Unable to access ..."
   }
}
```

The following is an example of when the Lambda function raises an overall `InternalError` exception, which is translated into a single time series with a status code of `InternalError` and a message. Whenever an error code has a value other than `Validation` or `Forbidden`, CloudWatch assumes that it's a generic internal error.

```
{
   "Error": {
      "Code": "PrometheusClusterUnreachable",
      "Value": "Unable to communicate with the cluster"
   }
}
```

DescribeGetMetricData event

Request payload

The following is an example of a `DescribeGetMetricData` request payload.

```
{
  "EventType": "DescribeGetMetricData"
}
```

Response payload

The following is an example of a `DescribeGetMetricData` response payload.

```
{
    "Description": "Data source connector",
    "ArgumentDefaults": [{
        Value: "default value"
```

```
    }]
}
```

- **Description**– A description of how to use the data source connector. This description will appear in the CloudWatch console. Markdown is supported.

  Type: String

- **ArgumentDefaults**– Optional array of argument default values used pre-populate the custom data source builder.

  If `[{ Value: "default value 1"}, { Value: 10}]`, is returned, the query builder in the CloudWatch console displays two inputs, the first with “default value 1” and the second with 10.

  If `ArgumentDefaults` is not provided, a single input is displayed with type default set to `String`.

  Type: Array of objects containing Value and Type.

- **Error**– (Optional) An error field can be included in any response. You can see examples in GetMetricData event.

Important considerations for CloudWatch alarms

If you are going to use the data source to set CloudWatch alarms, you should set it up to report data with timestamps every minute to CloudWatch. For more information and other considerations for creating alarms on metrics from connected data sources, see Create an alarm based on a connected data source.

(Optional) Use AWS Secrets Manager to store credentials

If your Lambda function needs to use credentials to access the data source, use AWS Secrets Manager to store these credentials instead of hardcoding them into your Lambda function. For more information about using AWS Secrets Manager with Lambda, see Use AWS Secrets Manager secrets in AWS Lambda functions.

(Optional) Connect to a data source in a VPC

If your data source is in a VPC managed by Amazon Virtual Private Cloud, you must configure your Lambda function to access it. For more information, see Connecting outbound networking to resources in a VPC.

You might also need to configure VPC service endpoints to access services such as AWS Secrets Manager. For more information, see Access an AWS service using an interface VPC endpoint.

Step 2: Create a Lambda permissions policy

You must use create a policy statement that grants CloudWatch permission to use the Lambda function that you created. You can use the AWS CLI or the Lambda console to create the policy statement.

To use the AWS CLI to create the policy statement

- Enter the following command. Replace *123456789012* with your account ID, replace *my-data-source-function* with the name of your Lambda function, and replace *MyDataSource-DataSourcePermission1234* with an arbitrary unique value.

```
aws lambda add-permission --function-name my-data-source-function --statement-
id MyDataSource-DataSourcePermission1234 --action lambda:InvokeFunction --principal
 lambda.datasource.cloudwatch.amazonaws.com --source-account 123456789012
```

Step 3: Attach a resource tag to the Lambda function

The CloudWatch console determines which of your Lambda functions are data source connectors by using a tag. When you create a data source using one of the wizards, the tag is automatically applied by the CloudFormation stack that configures it. When you create a data source yourself, you can use the following tag for your Lambda function. This makes your connector appear in the **Data source** dropdown in the CloudWatch console when you query metrics.

- A tag with `cloudwatch:datasource` as the key and `custom` as the value.

Use your custom data source

After you create a data source, you can use it to query and visualize data from that source, as well as to set alarms. If you used a template to create your custom data source connector or you added the tag listed in Step 3: Attach a resource tag to the Lambda function, you can follow the steps in Creating a graph of metrics from another data source to query it. You can also use the metric math function LAMBDA to query it, as explained in the following section. For information about creating alarms on metrics from your data source, see Create an alarm based on a connected data source. This topic describes how to pass arguments to your Lambda function to your custom data source.

How to pass arguments to your Lambda function

The recommended way for you to pass arguments to your custom data source is to use the query builder in the CloudWatch console when you query the data source.

You can also use your Lambda function to retrieve data from your data source by using the new LAMBDA expression in CloudWatch metric math.

```
LAMBDA("LambdaFunctionName" [, optional-arg]*)
```

`optional-arg` is up to 20 strings, numbers, or Booleans. For example, `param`, `3.14`, or `true`.

> **Note**
>
> Multi-line strings are not supported by the CloudWatch data source connectors. Every line feed is replaced with a space when the query is executed, or when you create an alarm or a dashboard widget with the query. In some cases, this might make your query not valid.

When you use the LAMBDA metric math function, you can provide the function name (`"MyFunction"`). If your resource policy allows, you can also use a specific version of the function (`"MyFunction:22"`), or a Lambda function alias (`"MyFunction:MyAlias"`). You can't use *

The following are some examples of calling the LAMBDA function.

```
LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query")
```

```
LAMBDA("MyCustomDataSource", true, "fuzzy", 99.9)
```

The LAMBDA metric math function returns a list of time series that can be returned to the requester or combined with other metric math functions. The following is an example of combining LAMBDA with other metric math functions.

```
FILL(LAMBDA("AmazonOpenSearchDataSource", "MyDomain", "some-query"), 0)
```

Delete a connector to a data source

The instructions in this section describe how to delete a connector to a data source.

To delete a connector to a data source

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Metrics data sources** tab.
4. Choose **Manage in CloudFormation** in the row of the data source that you want to delete.

   You are taken to the CloudFormation console.
5. In the section with the name of your data source, choose **Delete**.
6. In the confirmation pop-up, choose **Delete**.

Grafana integration

You can use Grafana version 6.5.0 and later to contextually advance through the CloudWatch console and query a dynamic list of metrics by using wildcards. This can help you monitor metrics for AWS resources, such as Amazon Elastic Compute Cloud instances or containers. When new instances are created as part of an Auto Scaling event, they appear in the graph automatically. You don't need to track the new instance IDs. Prebuilt dashboards help simplify the getting started experience for monitoring Amazon EC2, Amazon Elastic Block Store, and AWS Lambda resources.

You can use Grafana version 7.0 and later to perform CloudWatch Logs Insights queries on log groups in CloudWatch Logs. You can visualize your query results in bar, line, and stacked graphs and in a table format. For more information about CloudWatch Logs Insights, see Analyzing Log Data with CloudWatch Logs Insights.

For more information about how to get started, see Using AWS CloudWatch in Grafana in the Grafana Labs documentation.

AWS services that publish CloudWatch metrics

You can use the following table to learn which AWS services publish metrics to CloudWatch. For information about the metrics and dimensions, see the specified documentation.

Service	Namespace	Documentation
AWS Amplify	`AWS/AmplifyHosting`	Monitoring

Service	Namespace	Documentation
Amazon API Gateway	AWS/ApiGateway	Monitor API Execution with Amazon CloudWatch
Amazon AppFlow	AWS/AppFlow	Monitoring Amazon AppFlow with Amazon CloudWatch
AWS Application Migration Service	AWS/MGN	Monitoring Application Migration Service with Amazon CloudWatch
AWS App Runner	AWS/AppRunner	Viewing App Runner service metrics reported to CloudWatch
WorkSpaces Applications	AWS/AppStream	Monitoring Amazon AppStream 2.0 Resources
AWS AppSync	AWS/AppSync	CloudWatch Metrics
Amazon Athena	AWS/Athena	Monitoring Athena Queries with CloudWatch Metrics
Amazon Aurora	AWS/RDS	Amazon Aurora metrics
AWS Backup	AWS/Backup	Monitoring AWS Backup Metrics with CloudWatch
Amazon Bedrock Guardrails	AWS/Bedrock/Guardrails	Monitoring Amazon Bedrock Guardrails using CloudWatch Metrics
AWS Billing and Cost Management	AWS/Billing	Monitoring Charges with Alerts and Notifications
Amazon Braket	AWS/Braket/By Device	Monitoring Amazon Braket with Amazon CloudWatch
AWS Certificate Manager	AWS/CertificateManager	Supported CloudWatch Metrics

Service	Namespace	Documentation
AWS Private CA	`AWS/ACMPrivateCA`	Supported CloudWatch Metrics
Amazon Q Developer in chat applications	`AWS/Chatbot`	Monitoring Amazon Q Developer in chat applications with Amazon CloudWatch
Amazon Chime	`AWS/ChimeVoiceConnector`	Monitoring Amazon Chime with Amazon CloudWatch
Amazon Chime SDK	`AWS/ChimeSDK`	Service metrics
AWS Client VPN	`AWS/ClientVPN`	Monitoring with Amazon CloudWatch
Amazon CloudFront	`AWS/CloudFront`	Monitoring CloudFront Activity Using CloudWatch
AWS CloudHSM	`AWS/CloudHSM`	Getting CloudWatch Metrics
Amazon CloudSearch	`AWS/CloudSearch`	Monitoring an Amazon CloudSearch Domain with Amazon CloudWatch
AWS CloudTrail	`AWS/CloudTrail`	Supported CloudWatch metrics
CloudWatch agent	`CWAgent` or a custom namespace	Metrics collected by the CloudWatch agent
CloudWatch Application Signals	`ApplicationSignals`	Metrics collected by Application Signals

Service	Namespace	Documentation
CloudWatch metric streams	`AWS/CloudWatch/MetricStreams`	Monitoring your metric streams with CloudWatch metrics
CloudWatch RUM	`AWS/RUM`	CloudWatch metrics that you can collect with CloudWatch RUM
CloudWatch Synthetics	`CloudWatchSynthetics`	CloudWatch metrics published by canaries
Amazon CloudWatch Logs	`AWS/Logs`	Monitoring Usage with CloudWatch Metrics
AWS CodeBuild	`AWS/CodeBuild`	Monitoring AWS CodeBuild
Amazon CodeGuru Reviewer		Monitoring CodeGuru Reviewer with Amazon CloudWatch
AWS CodePipeline		AWS CodePipeline CloudWatch metrics
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch
Amazon Cognito	`AWS/Cognito`	Monitoring Amazon Cognito
Amazon Comprehend	`AWS/Comprehend`	Monitoring Amazon Comprehend endpoints
AWS Config	`AWS/Config`	AWS Config Usage and Success Metrics
Connect Customer	`AWS/Connect`	Monitoring Connect Customer in Amazon CloudWatch Metrics

Service	Namespace	Documentation
Amazon Data Lifecycle Manager	AWS/DataLifecycleManager	Monitor your policies using Amazon CloudWatch
AWS DataSync	AWS/DataSync	Monitoring Your Task
Amazon DataZone		Monitoring Amazon DataZone with Amazon CloudWatch
Amazon DevOps Guru	AWS/DevOps-Guru	Monitoring Amazon DevOps Guru with Amazon CloudWatch
AWS Database Migration Service	AWS/DMS	Monitoring AWS DMS Tasks
Direct Connect	AWS/DX	Monitoring with Amazon CloudWatch
AWS Directory Service	AWS/DirectoryService	Use Amazon CloudWatch metrics to determine when to add domain controllers
Amazon DocumentDB	AWS/DocDB	Monitoring Amazon DocumentDB with CloudWatch
Amazon DynamoDB	AWS/DynamoDB	DynamoDB Metrics and Dimensions
DynamoDB Accelerator (DAX)	AWS/DAX	Viewing DAX Metrics and Dimensions
Amazon EC2	AWS/EC2	Monitoring Your Instances Using CloudWatch
Amazon EC2 Elastic Graphics	AWS/ElasticGPUs	Using CloudWatch metrics to monitor Elastic Graphics

Service	Namespace	Documentation
Amazon EC2 Spot Fleet	AWS/EC2Spot	CloudWatch Metrics for Spot Fleet
Amazon EC2 Auto Scaling	AWS/AutoScaling	Monitoring Your Auto Scaling Groups and Instances Using CloudWatch
AWS Elastic Beanstalk	AWS/ElasticBeanstalk	Publishing Amazon CloudWatch Custom Metrics for an Environment
Amazon Elastic Block Store	AWS/EBS	Amazon CloudWatch Metrics for Amazon EBS
Amazon Elastic Container Registry	AWS/ECR	Amazon ECR repository metrics
Amazon Elastic Container Service	AWS/ECS	Amazon ECS CloudWatch Metrics
Amazon ECS through CloudWatch Container Insights	ECS/ContainerInsights	Amazon ECS Container Insights metrics
Amazon ECS Cluster auto scaling	AWS/ECS/ManagedScaling	Amazon ECS cluster auto scaling
AWS Elastic Disaster Recovery		CloudWatch Metrics for DRS
Amazon Elastic File System	AWS/EFS	Monitoring with CloudWatch

Service	Namespace	Documentation
Amazon Elastic Inference	AWS/ElasticInference	Using CloudWatch Metrics to Monitor Amazon Elastic Inference
Amazon EKS	AWS/EKS	Basic metrics in Amazon CloudWatch
Amazon EKS through CloudWatch Container Insights	ContainerInsights	Amazon EKS and Kubernetes Container Insights metrics
Elastic Load Balancing	AWS/ApplicationELB	CloudWatch Metrics for your Application Load Balancer
Elastic Load Balancing	AWS/NetworkELB	CloudWatch Metrics for your Network Load Balancer
Elastic Load Balancing	AWS/GatewayELB	CloudWatch Metrics for your Gateway Load Balancer
Elastic Load Balancing	AWS/ELB	CloudWatch Metrics for your Classic Load Balancer
Amazon ElastiCache (Memcached)	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon ElastiCache (Redis OSS)	AWS/ElastiCache	Monitoring Use with CloudWatch Metrics
Amazon OpenSearch Service	AWS/ES	Monitoring OpenSearch cluster metrics with Amazon CloudWatch

Service	Namespace	Documentation
Amazon EMR	AWS/ElasticMapReduce	Monitor Metrics with CloudWatch
Amazon EMR	AWS/EMRServerless	Monitoring EMR Serverless applications and jobs
AWS Elemental MediaConnect	AWS/MediaConnect	Monitoring MediaConnect with Amazon CloudWatch
AWS Elemental MediaConvert	AWS/MediaConvert	Using CloudWatch Metrics to View Metrics for AWS Elemental MediaConvert Resources
AWS Elemental MediaLive	AWS/MediaLive	Monitoring activity using Amazon CloudWatch metrics
AWS Elemental MediaPackage	AWS/MediaPackage	Monitoring AWS Elemental MediaPackage with Amazon CloudWatch Metrics
AWS Elemental MediaTailor	AWS/MediaTailor	Monitoring AWS Elemental MediaTailor with Amazon CloudWatch
AWS End User Messaging SMS	AWS/SMSVoice	Monitoring AWS End User Messaging SMS with Amazon CloudWatch
AWS End User Messaging Social	AWS/SocialMessaging	Monitoring AWS End User Messaging Social with Amazon CloudWatch
Amazon EventBridge	AWS/Events	Monitoring Amazon EventBridge
Amazon FinSpace		Logging and monitoring
Amazon Forecast		CloudWatch Metrics for Amazon Forecast

Service	Namespace	Documentation
Amazon Fraud Detector		Monitoring Amazon Fraud Detector with Amazon CloudWatch
Amazon FSx for Lustre	AWS/FSx	Monitoring Amazon FSx for Lustre
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon FSx for Windows File Server	AWS/FSx	Monitoring Amazon FSx for Windows File Server
Amazon FSx for NetApp ONTAP	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon FSx for OpenZFS	AWS/FSx	Monitoring with Amazon CloudWatch
Amazon GameLift Servers	AWS/GameLift	Monitor Amazon GameLift Servers with CloudWatch
AWS Global Accelerator	AWS/GlobalAccelerator	Using Amazon CloudWatch with AWS Global Accelerator
AWS Glue	Glue	Monitoring AWS Glue Using CloudWatch Metrics
AWS Ground Station	AWS/GroundStation	Metrics Using Amazon CloudWatch
AWS HealthLake	AWS/HealthLake	Monitoring HealthLake with CloudWatch
Amazon Inspector	AWS/Inspector	Monitoring Amazon Inspector Using CloudWatch

Service	Namespace	Documentation
Amazon Interactive Video Service	AWS/IVS	Monitoring Amazon IVS with Amazon CloudWatch
Amazon Interactive Video Service Chat	AWS/IVSChat	Monitoring Amazon IVS with Amazon CloudWatch
AWS IoT	AWS/IoT	AWS IoT Metrics and Dimensions
AWS IoT FleetWise	AWS/IoTFleetWise	Monitoring AWS IoT FleetWise with Amazon CloudWatch
AWS IoT SiteWise	AWS/IoTSiteWise	Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics
AWS IoT TwinMaker	AWS/IoTTwinMaker	Monitoring AWS IoT TwinMaker with Amazon CloudWatch metrics
AWS Key Management Service	AWS/KMS	Monitoring with CloudWatch
Amazon Keyspaces (for Apache Cassandra)	AWS/Cassandra	Amazon Keyspaces Metrics and Dimensions
Amazon Kendra		Monitoring Amazon Kendra with Amazon CloudWatch
Amazon Managed Service for Apache Flink	AWS/KinesisAnalytics	*Managed Service for Apache Flink for SQL Applications*: Monitoring with CloudWatch *Managed Service for Apache Flink for Apache Flink*: Viewing Amazon Managed Service for Apache Flink Metrics and Dimensions

Service	Namespace	Documentation
Amazon Data Firehose	`AWS/Firehose`	Monitoring Firehose Using CloudWatch Metrics
Amazon Kinesis Data Streams	`AWS/Kinesis`	Monitoring Amazon Kinesis Data Streams with Amazon CloudWatch
Amazon Kinesis Video Streams	`AWS/KinesisVideo`	Monitoring Kinesis Video Streams Metrics with CloudWatch
AWS Lambda	`AWS/Lambda`	AWS Lambda Metrics
Amazon Lex	`AWS/Lex`	Monitoring Amazon Lex with Amazon CloudWatch

AWS License Manager	`AWSLicenseManager/licenseUsage`
	
	`AWS/LicenseManager/LinuxSubscriptions`
		Monitoring license usage with Amazon CloudWatch
		
		Usage metrics and Amazon CloudWatch alarms for Linux subscriptions

Amazon Location Service	`AWS/Location`	Amazon Location Service metrics exported to Amazon CloudWatch
Amazon Lookout for Equipment	`AWS/lookoutequipment`	Monitoring Lookout for Equipment with Amazon CloudWatch
AWS Mainframe Modernization		Monitoring AWS Mainframe Modernization with Amazon CloudWatch
Amazon Machine Learning	`AWS/ML`	Monitoring Amazon ML with CloudWatch Metrics

Service	Namespace	Documentation
Amazon Managed Blockchain	AWS/managedblockchain	Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain
Amazon Managed Service for Prometheus	AWS/Prometheus	Amazon CloudWatch metrics
Amazon Managed Streaming for Apache Kafka	AWS/Kafka	Monitoring Amazon MSK with Amazon CloudWatch
Amazon Managed Streaming for Apache Kafka	AWS/KafkaConnect	Monitoring MSK Connect
Amazon Managed Workflows for Apache Airflow	AWS/MWAA	Container, queue, and database metrics for Amazon MWAA
Amazon MemoryDB	AWS/MemoryDB	Monitoring CloudWatch metrics
Amazon MQ	AWS/AmazonMQ	Monitoring Amazon MQ Brokers Using Amazon CloudWatch
Amazon Neptune	AWS/Neptune	Monitoring Neptune with CloudWatch
AWS Network Firewall	AWS/NetworkFirewall	AWS Network Firewall metrics in Amazon CloudWatch
AWS Network Manager	AWS/NetworkManager	CloudWatch metrics for on-premises resources

Service	Namespace	Documentation
AWS HealthOmics	AWS/Omics	Monitoring AWS HealthOmics with Amazon CloudWatch
AWS Outposts	AWS/Outposts	CloudWatch metrics for AWS Outposts
Amazon Personalize	AWS/Personalize	CloudWatch metrics for Amazon Personalize
Amazon Pinpoint	AWS/Pinpoint	View Amazon Pinpoint metrics in CloudWatch
Amazon Polly	AWS/Polly	Integrating CloudWatch with Amazon Polly
AWS PrivateLink	AWS/PrivateLinkEndpoints	CloudWatch metrics for AWS PrivateLink
AWS PrivateLink	AWS/PrivateLinkServices	CloudWatch metrics for AWS PrivateLink
AWS Private 5G	AWS/Private5G	Amazon CloudWatch metrics
Amazon Q Apps	AWS/QApps	Monitoring Amazon Q Business and Amazon Q Apps with Amazon CloudWatch
Amazon Q Business	AWS/QBusiness	Monitoring Amazon Q Business and Amazon Q Apps with Amazon CloudWatch
Amazon Q Developer	AWS/Q	Monitoring Amazon Q Developer with Amazon CloudWatch
Amazon Quick	AWS/QuickSight	Monitoring with Amazon CloudWatch

Service	Namespace	Documentation
Amazon Redshift	AWS/Redshift	Amazon Redshift Performance Data
Amazon Relational Database Service	AWS/RDS	Monitoring Amazon RDS metrics with Amazon CloudWatch
Amazon Rekognition	AWS/Rekognition	Monitoring Rekognition with Amazon CloudWatch
AWS re:Post Private	AWS/rePostPrivate	Monitoring AWS re:Post Private with Amazon CloudWatch
Amazon Route 53	AWS/Route53	Monitoring Amazon Route 53
Route 53 Application Recovery Controller	AWS/Route53RecoveryReadiness	Using Amazon CloudWatch with Application Recovery Controller
Amazon SageMaker AI	AWS/SageMaker	Monitoring SageMaker AI with CloudWatch
Amazon SageMaker AI Model Building Pipelines	AWS/SageMaker/ModelBuildingPipeline	SageMaker AI Pipelines Metrics
AWS Secrets Manager	AWS/SecretsManager	Monitoring Secrets Manager with Amazon CloudWatch
Amazon Security Lake	AWS/SecurityLake	CloudWatch metrics for Amazon Security Lake

Service	Namespace	Documentation
Service Catalog	AWS/ServiceCatalog	Service Catalog CloudWatch Metrics
AWS Shield Advanced	AWS/DDoSProtection	Monitoring with CloudWatch
Amazon Simple Email Service	AWS/SES	Retrieving Amazon SES Event Data from CloudWatch
Amazon Simple Notification Service	AWS/SNS	Monitoring Amazon SNS with CloudWatch
Amazon Simple Queue Service	AWS/SQS	Monitoring Amazon SQS Queues Using CloudWatch
Amazon S3	AWS/S3	Monitoring Metrics with Amazon CloudWatch
S3 Storage Lens	AWS/S3/Storage-Lens	Monitor S3 Storage Lens metrics in CloudWatch
Amazon Simple Workflow Service	AWS/SWF	Amazon SWF Metrics for CloudWatch
AWS Step Functions	AWS/States	Monitoring Step Functions Using CloudWatch
AWS Storage Gateway	AWS/StorageGateway	Using Amazon CloudWatch metrics
AWS Systems Manager Run Command	AWS/SSM-RunCommand	Monitoring Run Command Metrics Using CloudWatch
Amazon Textract	AWS/Textract	CloudWatch Metrics for Amazon Textract

Service	Namespace	Documentation
Amazon Timestream	AWS/Timestream	Timestream Metrics and Dimensions
AWS Transfer for SFTP	AWS/Transfer	AWS SFTP CloudWatch Metrics
Amazon Transcribe	AWS/Transcribe	Monitoring Amazon Transcribe with Amazon CloudWatch
Amazon Translate	AWS/Translate	CloudWatch Metrics and Dimensions for Amazon Translate
AWS Trusted Advisor	AWS/TrustedAdvisor	Creating Trusted Advisor Alarms Using CloudWatch
Amazon VPC	AWS/NATGateway	Monitoring Your NAT Gateway with CloudWatch
Amazon VPC	AWS/TransitGateway	CloudWatch Metrics for Your Transit Gateways
Amazon VPC	AWS/VPN	Monitoring with CloudWatch
Amazon VPC IP Address Manager	AWS/IPAM	Create alarms with Amazon CloudWatch
AWS WAF	AWS/WAFV2 for AWS WAF resources WAF for AWS WAF classic resources	Monitoring with CloudWatch
Amazon WorkMail	AWS/WorkMail	Monitoring Amazon WorkMail with Amazon CloudWatch

Service	Namespace	Documentation
Amazon WorkSpaces	`AWS/WorkSpaces`	Monitor Your WorkSpaces Using CloudWatch Metrics
Amazon WorkSpaces Web	`AWS/WorkSpacesWeb`	Monitoring Amazon WorkSpaces Web with Amazon CloudWatch

Using Amazon CloudWatch alarms

You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached. For example, you can monitor the CPU usage and disk reads and writes of your Amazon EC2 instances and then use that data to determine whether you should launch additional instances to handle increased load. You can also use this data to stop under-used instances to save money.

You can create both *metric* and *composite* alarms in Amazon CloudWatch.

You can create alarms on Metrics Insights queries that use AWS resource tags to filter and group metrics. To use tags with alarms, on the CloudWatch console, choose **Settings**. On the **CloudWatch Settings** page, under **Enable resource tags on telemetry**, choose **Enable**. For context-aware monitoring that automatically adapts to your tagging strategy, create alarms on Metrics Insights queries using AWS resource tags. This allows you to monitor all resources tagged with specific applications or environments.

- A *metric alarm* watches a single CloudWatch metric or the result of a math expression based on CloudWatch metrics. The alarm performs one or more actions based on the value of the metric or expression relative to a threshold over a number of time periods. The action can be sending a notification to an Amazon SNS topic, performing an Amazon EC2 action or an Amazon EC2 Auto Scaling action, starting an investigation in CloudWatch investigations, or creating an OpsItem or incident in Systems Manager.
- A *PromQL alarm* monitors metrics using a Prometheus Query Language (PromQL) instant query on metrics ingested through the CloudWatch OTLP endpoint. The alarm tracks individual breaching time series as contributors and uses duration-based pending and recovery periods to control state transitions. For more information, see PromQL alarms.
- A *log alarm* monitors the results of a CloudWatch Logs Insights query that runs on a schedule. The alarm evaluates aggregated query results against a threshold using M-out-of-N evaluation on recent query executions. Use log alarms to detect patterns, errors, or threshold breaches directly in your log data without creating metric filters. For more information, see Create a Log Alarm.
- A *composite alarm* includes a rule expression that takes into account the alarm states of other alarms that you have created. The composite alarm goes into ALARM state only if all conditions of the rule are met. The alarms specified in a composite alarm's rule expression can include metric alarms and other composite alarms.

Using composite alarms can reduce alarm noise. You can create multiple metric alarms, and also create a composite alarm and set up alerts only for the composite alarm. For example, a composite might go into ALARM state only when all of the underlying metric alarms are in ALARM state.

Composite alarms can send Amazon SNS notifications when they change state, and can create investigations, Systems Manager OpsItems, or incidents when they go into ALARM state, but can't perform EC2 actions or Auto Scaling actions.

> **Note**
>
> You can create as many alarms as you want in your AWS account.

You can add alarms to dashboards, so you can monitor and receive alerts about your AWS resources and applications across multiple regions. After you add an alarm to a dashboard, the alarm turns gray when it's in the `INSUFFICIENT_DATA` state and red when it's in the `ALARM` state. The alarm is shown with no color when it's in the `OK` state.

You also can favorite recently visited alarms from the *Favorites and recents* option in the CloudWatch console navigation pane. The *Favorites and recents* option has columns for your favorited alarms and recently visited alarms.

An alarm invokes actions only when the alarm changes state. The exception is for alarms with Auto Scaling actions. For Auto Scaling actions, the alarm continues to invoke the action once per minute that the alarm remains in the new state.

An alarm can watch a metric in the same account. If you have enabled cross-account functionality in your CloudWatch console, you can also create alarms that watch metrics in other AWS accounts. Creating cross-account composite alarms is not supported. Creating cross-account alarms that use math expressions is supported, except that the `ANOMALY_DETECTION_BAND`, `INSIGHT_RULE`, and `SERVICE_QUOTA` functions are not supported for cross-account alarms.

> **Note**
>
> CloudWatch doesn't test or validate the actions that you specify, nor does it detect any Amazon EC2 Auto Scaling or Amazon SNS errors resulting from an attempt to invoke nonexistent actions. Make sure that your alarm actions exist.

Concepts

CloudWatch alarms monitor metrics and trigger actions when thresholds are breached. Understanding how alarms evaluate data and respond to conditions is essential for effective monitoring.

Topics

- Alarm data queries
- Alarm evaluation
- PromQL alarms
- Log alarms
- Composite alarms
- Alarm actions
- Alarm Mute Rules
- Limits

Alarm data queries

CloudWatch alarms can monitor various data sources. Choose the appropriate query type based on your monitoring needs.

Metrics

Monitor a single CloudWatch metric. This is the most common alarm type for tracking resource performance. For more information about metrics, see CloudWatch Metrics concepts.

For more information, see Create a CloudWatch alarm based on a static threshold.

Metric math

You can set an alarm on the result of a math expression that is based on one or more CloudWatch metrics. A math expression used for an alarm can include as many as 10 metrics. Each metric must be using the same period.

For an alarm based on a math expression, you can specify how you want CloudWatch to treat missing data points. In this case, the data point is considered missing if the math expression doesn't return a value for that data point.

Alarms based on math expressions can't perform Amazon EC2 actions.

For more information about metric math expressions and syntax, see Math expressions with metrics.

For more information, see Create a CloudWatch alarm based on a metric math expression.

Metrics Insights

A CloudWatch Metrics Insights query helps you query metrics at scale using SQL-like syntax. You can create an alarm on any Metrics Insights query, including queries that return multiple time series. This capability significantly expands your monitoring options. When you create an alarm based on a Metrics Insights query, the alarm automatically adjusts as resources are added to or removed from your monitored group. Create the alarm once, and any resource that matches your query definition and filters joins the alarm monitoring scope when its corresponding metric becomes available. For multi-time series queries, each returned time series becomes a contributor to the alarm, allowing for more granular and dynamic monitoring.

Here are two primary use cases for CloudWatch Metrics Insights alarms:

- Anomaly Detection and Aggregate Monitoring

  Create an alarm on a Metrics Insights query that returns a single aggregated time series. This approach works well for dynamic alarms that monitor aggregated metrics across your infrastructure or applications. For example, you can monitor the maximum CPU utilization across all your instances, with the alarm automatically adjusting as you scale your fleet.

  To create an aggregate monitoring alarm, use this query structure:

```
SELECT FUNCTION(metricName)
```

```
                    FROM SCHEMA(...)
                    WHERE condition;
```

- Per-Resource Fleet Monitoring

  Create an alarm that monitors multiple time series, where each time series functions as a contributor with its own state. The alarm activates when any contributor enters the ALARM state, triggering resource-specific actions. For example, monitor database connections across multiple RDS instances to prevent connection rejections.

  To monitor multiple time series, use this query structure:

```
SELECT AVG(DatabaseConnections)
                    FROM AWS/RDS
                    WHERE condition
                    GROUP BY DBInstanceIdentifier
                    ORDER BY AVG() DESC;
```

  When creating multi-time series alarms, you must include two key clauses in your query:

  - A `GROUP BY` clause that defines how to structure the time series and determines how many time series the query will produce
  - An `ORDER BY` clause that establishes a deterministic sorting of your metrics, enabling the alarm to evaluate the most important signals first

  These clauses are essential for proper alarm evaluation. The `GROUP BY` clause splits your data into separate time series (for example, by instance ID), while the `ORDER BY` clause ensures consistent and prioritized processing of these time series during alarm evaluation.

For more information on how to create a multi time series alarm, see Create an alarm based on Metrics Insights.

Log group-metric filters

You can create an alarm based on a log group-metric filter. With metric filters, you can look for terms and patterns in log data as the data is sent to CloudWatch. For more information, see Create metrics from log events using filters in the *Amazon CloudWatch Logs User Guide*.

For more information on how to create an alarm based on log group-metric filter, see Alarming on logs.

Log Insights query

You can create an alarm that uses a CloudWatch Logs Insights query to monitor log data directly. The query runs on a schedule using a Scheduled Query, and the alarm evaluates the aggregated results against a threshold. When the aggregation expression includes a by clause, each unique combination of field values is tracked as a contributor.

For more information about how Log Alarms work, see Log alarms.

For more information about how to create a Log Alarm, see Create a Log Alarm.

PromQL

You can create an alarm that uses a Prometheus Query Language (PromQL) instant query to monitor metrics ingested through the CloudWatch OTLP endpoint.

For more information about how PromQL alarms work, see PromQL alarms.

For more information on how to create a PromQL alarm, see Create an alarm using a PromQL query.

External data source

You can create alarms that watch metrics from data sources that aren't in CloudWatch. For more information about creating connections to these other data sources, see Query metrics from other data sources.

For more information on how to create an alarm based on a connected data source, see Create an alarm based on a connected data source.

Alarm evaluation

Metric alarm states

A metric alarm has the following possible states:

- `OK` – The metric or expression is within the defined threshold.
- `ALARM` – The metric or expression is outside of the defined threshold.
- `INSUFFICIENT_DATA` – The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.

Alarm evaluation state

In addition to the alarm state, each alarm has an evaluation state that provides information about the alarm evaluation process. The following states may occur:

- `PARTIAL_DATA` – Indicates that not all the available data was able to be retrieved due to quota limitations. For more information, see How partial data is handled.
- `IN_WARM_UP` – Indicates that the alarm is in its warm-up period and does not yet evaluate. The alarm remains in `INSUFFICIENT_DATA` and does not perform alarm actions until the warm-up period ends. For more information, see Alarm warm-up periods.
- `EVALUATION_ERROR` – Indicates configuration errors in alarm setup that require review and correction. Refer to StateReason field of the alarm for more details.
- `EVALUATION_FAILURE` – Indicates temporary CloudWatch issues. We recommend manual monitoring until the issue is resolved

You can view the evaluation state in the alarm details in the console, or by using the `describe-alarms` CLI command or `DescribeAlarms` API.

Alarm evaluation settings

When you create an alarm, you specify 3 settings to enable CloudWatch to evaluate when to change the alarm state:

- **Period** is the length of time to use to evaluate the metric or expression to create each individual data point for an alarm. It is expressed in seconds.
- **Evaluation Periods** is the number of the most recent periods, or data points, to evaluate when determining alarm state.
- **Datapoints to Alarm** is the number of data points within the Evaluation Periods that must be breaching to cause the alarm to go to the `ALARM` state. The breaching data points don't have to be consecutive, but they must all be within the last number of data points equal to **Evaluation Period**.

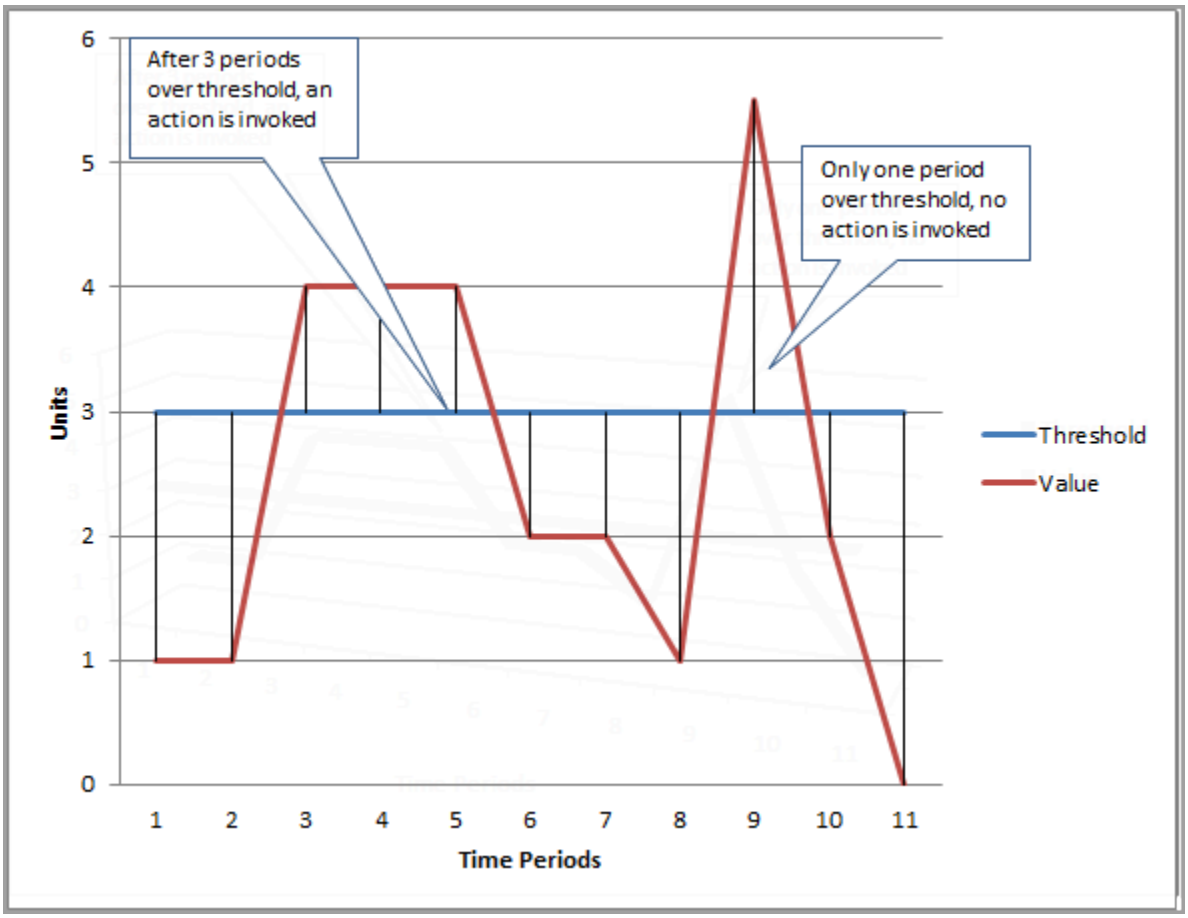
For any period of one minute or longer, an alarm is evaluated every minute and the evaluation is based on the window of time defined by the **Period** and **Evaluation Periods**. For example, if the **Period** is 5 minutes (300 seconds) and **Evaluation Periods** is 1, then at the end of minute 5

the alarm evaluates based on data from minutes 1 to 5. Then at the end of minute 6, the alarm is evaluated based on the data from minutes 2 to 6.

If the alarm period is 10 seconds, 20 seconds, or 30 seconds, the alarm is evaluated every 10 seconds. For more information, see High-resolution alarms.

If the number of evaluation periods multiplied by the length of each evaluation period exceeds one day, the alarm is evaluated once per hour. For more details about how these multi-day alarms are evaluated, see Example of evaluating a multi-day alarm.

In the following figure, the alarm threshold for a metric alarm is set to 3 units. Both **Evaluation Period** and **Datapoints to Alarm** are 3. That is, when all existing data points in the most recent 3 consecutive periods are above the threshold, the alarm goes to ALARM state. In the figure, this happens in the third through fifth time periods. At period six, the value dips below the threshold, so one of the periods being evaluated is not breaching, and the alarm state changes back to OK. During the ninth time period, the threshold is breached again, but for only 1 period. Consequently, the alarm state remains OK.



When you configure **Evaluation Periods** and **Datapoints to Alarm** as different values, you're setting an "M out of N" alarm. **Datapoints to Alarm** is ("M") and **Evaluation Periods** is ("N").The evaluation interval is the number of evaluation periods multiplied by the period length. For example, if you configure 4 out of 5 data points with a period of 1 minute, the evaluation interval is 5 minutes. If you configure 3 out of 3 data points with a period of 10 minutes, the evaluation interval is 30 minutes.

> **Note**
>
> If data points are missing soon after you create an alarm, and the metric was being reported to CloudWatch before you created the alarm, CloudWatch retrieves the most recent data points from before the alarm was created when evaluating the alarm.

By default, CloudWatch uses a *sliding window* for alarm evaluation. Each time the alarm is evaluated, the window advances by one minute, and the boundaries of the window are not aligned to the wall clock. You can instead align the evaluation window to wall clock boundaries and time zones. For more information, see Alarm evaluation window.

High-resolution alarms

If you set an alarm on a high-resolution metric, you can specify a high-resolution alarm with a period of 10 seconds, 20 seconds, or 30 seconds. There is a higher charge for high-resolution alarms. For more information about high-resolution metrics, see Publish custom metrics (PutMetricData / EMF).

Example of evaluating a multi-day alarm

An alarm is a multi-day alarm if the number of evaluation periods multiplied by the length of each evaluation period exceeds one day. Multi-day alarms are evaluated once per hour. When multi-day alarms are evaluated, CloudWatch takes into account only the metrics up to the current hour at the :00 minute when evaluating. This top-of-the-hour behavior, and the following example, apply to alarms that use a sliding evaluation window, which is the default.

For example, consider an alarm that monitors a job that runs every 3 days at 10:00.

1. At 10:02, the job fails
2. At 10:03, the alarm evaluates and stays in OK state, because the evaluation considers data only up to 10:00.

3. At 11:03, the alarm considers data up to 11:00 and goes into ALARM state.
4. At 11:43, you correct the error and the job now runs successfully.
5. At 12:03, the alarm evaluates again, sees the successful job, and returns to OK state.

If the alarm instead uses a wall clock evaluation window, CloudWatch aligns the evaluation window to the wall clock boundary that matches the period in the alarm's selected time zone, rather than clamping to the top of each hour. For example, a multi-day alarm with a one-day period evaluates complete calendar days that end at midnight in the selected time zone, and an alarm with a one-week period evaluates complete calendar weeks. The window advances by one full period when each boundary is reached, so the alarm considers a newly completed calendar period only after that period ends. For more information, see Alarm evaluation window.

Alarm evaluation window

When CloudWatch evaluates a metric alarm, it retrieves metric data for a time range called the *evaluation window*. The length of the window is `Period` multiplied by `Evaluation Periods`. Where the boundaries of that window fall is controlled by the `EvaluationWindow` parameter on the PutMetricAlarm API. `EvaluationWindow` accepts one of two configurations:

- `SlidingWindow` – the default. The window advances each time the alarm is evaluated, forming a rolling time window.
- `WallClockWindow` – the window is aligned to clock boundaries that match the period (such as the top of the hour, midnight, or the start of the calendar week) and, optionally, to a specific time zone.

If you don't specify `EvaluationWindow` when you create or update a metric alarm, the alarm uses a sliding window. You can switch an existing alarm to a wall clock window at any time by updating the alarm.

Both configurations are supported on all types of metric alarms except alarms that are based on a PromQL query.

Choosing between a sliding window and a wall clock window

A sliding window continuously incorporates the most recent data, so it detects breaches as soon as data points cross your threshold. Choose a sliding window for real-time monitoring where calendar alignment doesn't matter, such as performance, latency, or resource-exhaustion workloads.

A wall clock window evaluates complete, calendar-aligned periods such as each hour, day, or week. Choose a wall clock window when monitoring is tied to a business or calendar period. It also avoids false alarms that can occur when events span rolling window boundaries. For example, a daily backup alarm with a sliding window can enter `ALARM` if consecutive backups are more than 24 hours apart, even though each calendar day had a backup. A wall clock window evaluates each day independently and avoids this.

The trade-off is detection timing. A wall clock window reflects a data point only after the period that contains it ends. If you need to detect a problem while it is still in progress, use a sliding window.

Sliding window

With a sliding window, the evaluation window moves forward each time the alarm is evaluated, forming a rolling time window. For example, an alarm with a 1-hour period and 1 evaluation period might evaluate the time range 13:07–14:07 on one evaluation and 13:08–14:08 on the next.

Because the window always incorporates the most recent data points, a sliding window is the better choice for alarms that trigger Auto Scaling actions.

A sliding window has no additional configuration options. To configure one explicitly through the API, set `EvaluationWindow` to `{"SlidingWindow":{}}`. Omitting `EvaluationWindow` has the same effect.

Wall clock window

Evaluation periods align to fixed time boundaries (for example, each hour or day) in your selected time zone. Best for scheduled workloads like daily reports, batch jobs, or backups.

Boundaries align to the start of the matching calendar period in the selected time zone: the top of the hour for a 1-hour period, midnight for a 1-day period, and Monday at 00:00 (the start of the ISO 8601 week) for a 1-week period. These alignment points are fixed and can't be changed.

When to use a wall clock window

Choose a wall clock window when you want alarm evaluations to align with time-of-day or calendar boundaries. Common scenarios include:

- Daily batch jobs that should complete by midnight in a specific time zone.
- Weekly compliance or business reports that summarize a calendar week.

> **Note**
>
> We don't recommend using a wall clock window for alarms that trigger Auto Scaling actions. The alarm reflects new data only after the current period ends, so it can't track load closely enough for responsive scaling. Use a sliding window instead.

Supported periods

You can use a wall clock window with metric alarms that have one of the following periods:

- 1 minute (60 seconds)
- 5 minutes (300 seconds)
- 1 hour (3600 seconds)
- 1 day (86400 seconds)
- 1 week (604800 seconds)

Wall clock windows are not supported for high-resolution alarms (periods of 10, 20, or 30 seconds).

If you specify a period that wall clock windows don't support, `PutMetricAlarm` returns a validation error that lists the supported periods.

For metric math and Metrics Insights alarms, every metric in the `Metrics` array must use one of the supported wall clock periods.

> **Note**
>
> A Metrics Insights alarm can evaluate a time range of no more than 3 hours (**Period** multiplied by **Evaluation Periods**). When a Metrics Insights alarm uses a wall clock window, CloudWatch automatically reserves one additional period to align the window to wall clock boundaries. As a result, the effective limit becomes **Period** multiplied by (**Evaluation Periods** + 1), which must still be 3 hours or less. This means a Metrics Insights alarm that uses a wall clock window can span one fewer evaluation period than the same alarm with a sliding window. For example, with a 1-hour period, a sliding window supports up to 3 evaluation periods, but a wall clock window supports up to 2. If you exceed this limit, `PutMetricAlarm` returns a validation error.

Time zones and daylight saving time

You can optionally specify a time zone for the wall clock window. If you don't specify a time zone, CloudWatch uses UTC. The time zone determines where wall clock boundaries fall: for example, an alarm with a 1-day period in `America/New_York` evaluates each calendar day from local midnight to local midnight in New York, while the same alarm in UTC evaluates from 00:00 UTC to 00:00 UTC. Time zones with non-whole-hour offsets such as `Asia/Kolkata` (`+05:30`) shift the boundaries even for hourly and sub-hourly periods.

You can specify the time zone in any of the following forms:

- An IANA time zone identifier, such as `America/New_York`, `Europe/London`, or `Asia/Kolkata`.
- A fixed UTC offset, such as `+05:30`, `-08:00`, or `Z` (equivalent to UTC).
- An offset-prefixed identifier, such as `UTC+05:30` or `GMT-08:00`.

The time zone you specify must resolve to a UTC offset that is a multiple of 5 minutes. For example, `+01:00`, `+05:30`, and `+01:05` are accepted, but `+01:03` is rejected. CloudWatch rejects time zones that don't meet this requirement with a validation error at alarm creation.

When you specify an IANA time zone, CloudWatch automatically handles daylight saving time transitions. The evaluation window remains aligned to the local wall clock in the specified time zone, even on the days when clocks change. Fixed UTC offsets and offset-prefixed identifiers don't observe daylight saving time.

How often the alarm is evaluated

An alarm with a wall clock window is evaluated at the same frequency as an alarm with a sliding window. Choosing a wall clock window changes only the *boundaries* of the window that CloudWatch uses to retrieve metric data, not how often the alarm runs.

For example, an alarm with a 1-hour period and a wall clock window evaluates every minute. Each evaluation looks at the data for the most recently completed clock hour. As a new clock hour begins, the window advances to the new hour boundary.

To create a metric alarm that uses a wall clock window, see Create a metric alarm that uses a wall clock evaluation window.

Alarm warm-up periods

A warm-up period delays alarm evaluation after you create or update an alarm. This reduces noise from missing data while a new resource or service starts to publish data. During warm-up, the alarm stays in `INSUFFICIENT_DATA` and does not perform alarm actions.

Use a warm-up period when you deploy alarms with a new application or service that takes time to emit metrics. Without a warm-up period, the alarm can transition and run actions on missing data during startup, and cause false notifications.

Warm-up periods work with metric alarms and log alarms.

Warm-up period configuration

Set the `WarmUpConfiguration` parameter when you create or update an alarm. This parameter includes the following settings:

- `WarmUpPeriodDurationInMinutes` (required) – The warm-up duration, in minutes. Specify a value from 1 to 2,880 minutes (2 days).
- `OnlyStartEvaluatingAfterWarmUpPeriodEnds` (optional, default is `false`) – Controls when evaluation begins. When set to `false` (default), the alarm ends warm-up early. Evaluation begins as soon as enough data fills its evaluation window. When set to `true`, the alarm waits the full warm-up duration before it evaluates, even if data arrives sooner. Use this setting when startup fluctuations, such as a CPU spike, could cause premature alarm transitions.

When `OnlyStartEvaluatingAfterWarmUpPeriodEnds` is `false`, the alarm ends warm-up as soon as it can fill its evaluation window. The evaluation window spans `Period` multiplied by `Evaluation Periods` and is full when a datapoint exists for the most recent period.

For example, an alarm with a 5-minute period and 3 evaluation periods has a 15-minute evaluation window. Suppose that you create this alarm at 10:00 and the resource starts to publish at 10:04. The alarm stays in `INSUFFICIENT_DATA` until a full 15-minute window is available. This happens at about 10:19, and then the alarm begins to evaluate. The `Datapoints to Alarm` setting does not affect when warm-up ends.

Change or end a warm-up period

You can change the warm-up configuration while the alarm is still in its warm-up period. You can also end warm-up early when you update the configuration. Warm-up applies only once at alarm creation and does not restart.

After the warm-up period ends, you can update the warm-up configuration. However, the update does not start a new warm-up period. The alarm continues to evaluate normally.

View warm-up status

During warm-up, the alarm state remains `INSUFFICIENT_DATA` and the evaluation state is `IN_WARM_UP`. View the alarm configuration and state with the DescribeAlarms API. For more information about evaluation states, see Alarm evaluation state.

For instructions on creating an alarm with a warm-up period, see Create an alarm that uses a warm-up period.

Configuring how CloudWatch alarms treat missing data

Sometimes, not every expected data point for a metric gets reported to CloudWatch. For example, this can happen when a connection is lost, a server goes down, or when a metric reports data only intermittently by design.

CloudWatch enables you to specify how to treat missing data points when evaluating an alarm. This helps you to configure your alarm so that it goes to `ALARM` state only when appropriate for the type of data being monitored. You can avoid false positives when missing data doesn't indicate a problem.

Similar to how each alarm is always in one of three states, each specific data point reported to CloudWatch falls under one of three categories:

- Not breaching (within the threshold)
- Breaching (violating the threshold)
- Missing

For each alarm, you can specify CloudWatch to treat missing data points as any of the following:

- `notBreaching` – Missing data points are treated as "good" and within the threshold
- `breaching` – Missing data points are treated as "bad" and breaching the threshold
- `ignore` – The current alarm state is maintained
- `missing` – If all data points in the alarm evaluation range are missing, the alarm transitions to INSUFFICIENT_DATA.

The best choice depends on the type of metric and the purpose of the alarm. For example, if you are creating an application rollback alarm using a metric that continually reports data, you might want to treat missing data points as breaching, because it might indicate that something is wrong. But for a metric that generates data points only when an error occurs, such as `ThrottledRequests` in Amazon DynamoDB, you would want to treat missing data as `notBreaching`. The default behavior is `missing`.

> **Important**
>
> Alarms configured on Amazon EC2 metrics can temporarily enter the INSUFFICIENT_DATA state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions, we recommend that you configure those alarms to treat missing data as `missing`, and to have these alarms trigger only when in the ALARM state.

Choosing the best option for your alarm prevents unnecessary and misleading alarm condition changes, and also more accurately indicates the health of your system.

> **Important**
>
> Alarms that evaluate metrics in the `AWS/DynamoDB` namespace default to ignore missing data. You can override this if you choose a different option for how the alarm should treat missing data. When an `AWS/DynamoDB` metric has missing data, alarms that evaluate that metric remain in their current state.

How alarm state is evaluated when data is missing

Whenever an alarm evaluates whether to change state, CloudWatch attempts to retrieve a higher number of data points than the number specified as **Evaluation Periods**. The exact number of data points it attempts to retrieve depends on the length of the alarm period and whether it is based on a metric with standard resolution or high resolution. The time frame of the data points that it attempts to retrieve is the *evaluation range*.

> **Note**
>
> This wider evaluation range applies only to alarms that use a sliding evaluation window, which is the default. Alarms that use a wall clock evaluation window retrieve exactly the data points in the aligned window (**Period** multiplied by **Evaluation Periods**) and do not query additional data points from farther back. For more information, see Alarm evaluation window.

Once CloudWatch retrieves these data points, the following happens:

- If no data points in the evaluation range are missing, CloudWatch evaluates the alarm based on the most recent data points collected. The number of data points evaluated is equal to the **Evaluation Periods** for the alarm. The extra data points from farther back in the evaluation range are not needed and are ignored.
- If some data points in the evaluation range are missing, but the total number of existing data points that were successfully retrieved from the evaluation range is equal to or more than the alarm's **Evaluation Periods**, CloudWatch evaluates the alarm state based on the most recent real data points that were successfully retrieved, including the necessary extra data points from farther back in the evaluation range. In this case, the value you set for how to treat missing data is not needed and is ignored.
- If some data points in the evaluation range are missing, and the number of actual data points that were retrieved is lower than the alarm's number of **Evaluation Periods**, CloudWatch fills in the missing data points with the result you specified for how to treat missing data, and then evaluates the alarm. However, all real data points in the evaluation range are included in the evaluation. CloudWatch uses missing data points only as few times as possible.

> **Note**
>
> A particular case of this behavior is that CloudWatch alarms might repeatedly re-evaluate the last set of data points for a period of time after the metric has stopped flowing. This re-evaluation might cause the alarm to change state and re-execute actions, if it had changed state immediately before the metric stream stopping. To mitigate this behavior, use shorter periods.

The following tables illustrate examples of the alarm evaluation behavior. In the first table, **Datapoints to Alarm** and **Evaluation Periods** are both 3. CloudWatch retrieves the 5 most recent data points when evaluating the alarm, in case some of the most recent 3 data points are missing. 5 is the evaluation range for the alarm.

Column 1 shows the 5 most recent data points, because the evaluation range is 5. These data points are shown with the most recent data point on the right. 0 is a non-breaching data point, X is a breaching data point, and - is a missing data point.

Column 2 shows how many of the 3 necessary data points are missing. Even though the most recent 5 data points are evaluated, only 3 (the setting for **Evaluation Periods**) are necessary to evaluate the alarm state. The number of data points in Column 2 is the number of data points that must be "filled in", using the setting for how missing data is being treated.

In columns 3-6, the column headers are the possible values for how to treat missing data. The rows in these columns show the alarm state that is set for each of these possible ways to treat missing data.

Data points	# of data points that must be filled	Missing	Ignore	Breaching	Not breaching
0 - X - X	0	OK	OK	OK	OK
0 - - - -	2	OK	OK	OK	OK
- - - - -	3	INSUFFICIENT_DATA	Retain current state	ALARM	OK
0 X X - X	0	ALARM	ALARM	ALARM	ALARM
- - X - -	2	ALARM	Retain current state	ALARM	OK

In the second row of the preceding table, the alarm stays OK even if missing data is treated as breaching, because the one existing data point is not breaching. This is evaluated along with two missing data points which are treated as breaching. The next time this alarm is evaluated, if the

data is still missing it will go to ALARM, as that non-breaching data point will no longer be in the evaluation range.

The third row, where all five of the most recent data points are missing, illustrates how the various settings for how to treat missing data affect the alarm state. If missing data points are considered breaching, the alarm goes into ALARM state, while if they are considered not breaching, then the alarm goes into OK state. If missing data points are ignored, the alarm retains the current state it had before the missing data points. And if missing data points are just considered as missing, then the alarm does not have enough recent real data to make an evaluation, and goes into INSUFFICIENT_DATA.

In the fourth row, the alarm goes to ALARM state in all cases because the three most recent data points are breaching. The alarm's **Evaluation Periods** and **Datapoints to Alarm** are both set to 3. In this case, the missing data point is ignored and the setting for how to evaluate missing data is not needed, because there are 3 real data points to evaluate.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see Avoiding premature transitions to alarm state.

In the next table, the **Period** is again set to 5 minutes, and **Datapoints to Alarm** is only 2 while **Evaluation Periods** is 3. This is a 2 out of 3, M out of N alarm.

The evaluation range is 5. This is the maximum number of recent data points that are retrieved and can be used in case some data points are missing.

Data points	# of missing data points	Missing	Ignore	Breaching	Not breaching
0 - X - X	0	ALARM	ALARM	ALARM	ALARM
0 0 X 0 X	0	ALARM	ALARM	ALARM	ALARM
0 - X - -	1	OK	OK	ALARM	OK
- - - - 0	2	OK	OK	ALARM	OK
- - - X -	2	ALARM	Retain current state	ALARM	OK

In rows 1 and 2, the alarm always goes to ALARM state because 2 of the 3 most recent data points are breaching. In row 2, the two oldest data points in the evaluation range are not needed because none of the 3 most recent data points are missing, so these two older data points are ignored.

In rows 3 and 4, the alarm goes to ALARM state only if missing data is treated as breaching, in which case the two most recent missing data points are both treated as breaching. In row 4, these two missing data points that are treated as breaching provide the two necessary breaching data points to trigger the ALARM state.

Row 5 represents a special case of alarm evaluation called *premature alarm state*. For more information, see the following section.

Avoiding premature transitions to alarm state

CloudWatch alarm evaluation includes logic to try to avoid false alarms, where the alarm goes into ALARM state prematurely when data is intermittent. The example shown in row 5 in the tables in the previous section illustrate this logic. In those rows, and in the following examples, the **Evaluation Periods** is 3 and the evaluation range is 5 data points. **Datapoints to Alarm** is 3, except for the M out of N example, where **Datapoints to Alarm** is 2.

Suppose an alarm's most recent data is - - - - X, with four missing data points and then a breaching data point as the most recent data point. Because the next data point may be non-breaching, the alarm does not go immediately into ALARM state when the data is either - - - - X or - - - X - and **Datapoints to Alarm** is 3. This way, false positives are avoided when the next data point is non-breaching and causes the data to be - - - X 0 or - - X - 0.

However, if the last few data points are - - X - -, the alarm goes into ALARM state even if missing data points are treated as missing. This is because alarms are designed to always go into ALARM state when the oldest available breaching datapoint during the **Evaluation Periods** number of data points is at least as old as the value of **Datapoints to Alarm**. All other more recent data points are breaching or missing. In this case, the alarm goes into ALARM state even if the total number of datapoints available is lower than M (**Datapoints to Alarm**).

This alarm logic applies to M out of N alarms as well. If the oldest breaching data point during the evaluation range is at least as old as the value of **Datapoints to Alarm**, and all of the more recent data points are either breaching or missing, the alarm goes into ALARM state no matter the value of M (**Datapoints to Alarm**).

Missing Data in CloudWatch Metrics Insights alarms

Alarms based on Metrics Insights queries that aggregate to a single time series

The missing data scenarios and their effects upon alarm evaluation are the same as a standard metric alarm in terms of the configured missing data treatment. See, Configuring how CloudWatch alarms treat missing data.

Alarms based on Metrics Insights queries that produce multiple time series

Missing data scenarios for Metrics Insights alarms occur when:

- Individual datapoints within a time series are not present.
- One or more time series disappear when evaluating upon multiple time series.
- No time series are retrieved by the query.

Missing data scenarios affect the alarm evaluation in the following manner:

- For the evaluation of a time series, the treat missing data treatment is applied for individual datapoints within the time series. For example, if 3 datapoints were queried for the time series but only 1 was received, 2 datapoints would follow the configured missing data configuration.
- If a time series is not retrieved by the query anymore, it will transition to `OK` no matter the treat missing data treatment. Alarm actions associated with the `OK` transition at the contributor level are executed and the `StateReason` specifies that the aforementioned contributor was not found with the message, "No data was returned for this contributor". The state of the alarm will depend on the state of the other contributors that were retrieved by the query.
- At alarm level, if the query returns an empty result (no time series at all), the treat missing data treatment is applied. For example, if the treat missing data was set as `BREACHING`, the alarm will transition to `ALARM`.

How partial data is handled

How partial data from a Metrics Insights query is evaluated

If the Metrics Insights query used for the alarm matches more than 10,000 metrics, the alarm is evaluated based on the first 10,000 metrics that the query finds. This means that the alarm is being evaluated on partial data.

You can use the following methods to find whether a Metrics Insights alarm is currently evaluating its alarm state based on partial data:

- In the console, if you choose an alarm to see the **Details** page, the message **Evaluation warning: Not evaluating all data** appears on that page.
- You see the value `PARTIAL_DATA` in the `EvaluationState` field when you use the describe-alarms AWS CLI command or the DescribeAlarms API.

Alarms also publish events to Amazon EventBridge when it goes into the partial data state, so you can create an EventBridge rule to watch for these events. In these events, the `evaluationState` field has the value `PARTIAL_DATA`. The following is an example.

```
{
    "version": "0",
    "id": "12345678-3bf9-6a09-dc46-12345EXAMPLE",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-11-08T11:26:05Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:my-alarm-name"
    ],
    "detail": {
        "alarmName": "my-alarm-name",
        "state": {
            "value": "ALARM",
            "reason": "Threshold Crossed: 3 out of the last 3 datapoints [20000.0
 (08/11/22 11:25:00), 20000.0 (08/11/22 11:24:00), 20000.0 (08/11/22 11:23:00)] were
 greater than the threshold (0.0) (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2022-11-08T11:26:05.399+0000\",\"startDate\":\"2022-11-08T11:23:00.000+0000\",
\"period\":60,\"recentDatapoints\":[20000.0,20000.0,20000.0],\"threshold\":0.0,
\"evaluatedDatapoints\":[{\"timestamp\":\"2022-11-08T11:25:00.000+0000\",\"value
\":20000.0}]}",
            "timestamp": "2022-11-08T11:26:05.401+0000",
            "evaluationState": "PARTIAL_DATA"
        },
        "previousState": {
            "value": "INSUFFICIENT_DATA",
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2022-11-08T11:25:51.227+0000"
        },
        "configuration": {
```

```
            "metrics": [
                {
                    "id": "m2",
                    "expression": "SELECT SUM(PartialDataTestMetric) FROM
 partial_data_test",
                    "returnData": true,
                    "period": 60
                }
            ]
        }
    }
}
```

If the query for the alarm includes a GROUP BY statement that initially returns more than 500 time series, the alarm is evaluated based on the first 500 time series that the query finds. However, if you use an ORDER BY clause, then all the time series that the query finds are sorted. The 500 that have the highest or lowest values according to your ORDER BY clause are used to evaluate the alarm.

How partial data from a multi data source alarm is evaluated

If the Lambda function returns partial data:

- The alarm continues to be evaluated on the data points that are returned.
- You can use the following methods to find whether an alarm on a Lambda function is currently evaluating its alarm state based on partial data:
  - In the console, choose an alarm and choose the **Details** page. If you see the message **Evaluation warning: Not evaluating all data appears on that page**, it is evaluating on partial data.
  - If you see the value `PARTIAL_DATA` in the `EvaluationState` field when you use the `describe-alarms` AWS CLI command or the DescribeAlarms API, it is evaluating on partial data.
- An alarm also publishes events to Amazon EventBridge when it goes into the partial data state.

Percentile-based alarms and low data samples

When you set a percentile as the statistic for an alarm, you can specify what to do when there is not enough data for a good statistical assessment. You can choose to have the alarm evaluate the statistic anyway and possibly change the alarm state. Or, you can have the alarm ignore the metric

while the sample size is low, and wait to evaluate it until there is enough data to be statistically significant.

For percentiles between 0.5 (inclusive) and 1.00 (exclusive), this setting is used when there are fewer than 10/(1-percentile) data points during the evaluation period. For example, this setting would be used if there were fewer than 1000 samples for an alarm on a p99 percentile. For percentiles between 0 and 0.5 (exclusive), the setting is used when there are fewer than 10/percentile data points.

PromQL alarms

A PromQL alarm monitors metrics using a Prometheus Query Language (PromQL) instant query. The query selects metrics ingested through the CloudWatch OTLP endpoint, and all matching time series returned by the query are considered to be breaching. The alarm evaluates the query at a regular interval and tracks each breaching time series independently as a *contributor*.

For information about ingesting metrics using OpenTelemetry, see OpenTelemetry.

How PromQL alarms work

A PromQL alarm evaluates a PromQL instant query on a recurring schedule defined by the `EvaluationInterval`. The query returns only the time series that satisfy the condition. Each returned time series is a *contributor*, identified by its unique set of attributes.

The alarm uses duration-based state transitions:

- When a contributor is returned by the query, it is considered *breaching*. If the contributor continues breaching for the duration specified by `PendingPeriod`, the contributor transitions to `ALARM` state.
- When a contributor stops being returned by the query, it is considered *recovering*. If the contributor remains absent for the duration specified by `RecoveryPeriod`, the contributor transitions back to `OK` state.

The alarm is in `ALARM` state when at least one contributor has been breaching for longer than the pending period. The alarm returns to `OK` state when all contributors have recovered.

> **Note**
>
> You can reference a PromQL alarm as a child alarm in a CloudWatch composite alarm's rule expression, the same as any other alarm type. For more information, see Create a composite alarm.

PromQL alarm configuration

A PromQL alarm is configured with the following parameters:

- **PendingPeriod** is the duration in seconds that a contributor must continuously breach before the contributor transitions to `ALARM` state. This is equivalent to the Prometheus alert rule's `for` duration.
- **RecoveryPeriod** is the duration in seconds that a contributor must stop breaching before the contributor transitions back to `OK` state. This is equivalent to the Prometheus alert rule's `keep_firing_for` duration.
- **EvaluationInterval** is how frequently, in seconds, the alarm evaluates the PromQL query.

To create a PromQL alarm, see Create an alarm using a PromQL query.

Log alarms

A Log Alarm monitors the results of a CloudWatch Logs Insights query that runs on a schedule using a Scheduled Query. The alarm applies an aggregation expression to the query results to produce a numeric value, and when that aggregated value breaches a configured threshold, the alarm transitions to `ALARM` state and runs configured actions.

Unlike metric alarms that require metric filters as an intermediate step, Log Alarms evaluate directly on log data using the same Logs Insights query language you use for ad-hoc analysis.

How Log Alarms work

The following steps describe how a Log Alarm works:

1. You create a Log Alarm with a query, aggregation expression, schedule, and threshold.
2. CloudWatch automatically creates an AWS managed Scheduled Query that runs your query on the specified schedule.

3. Each query execution produces aggregated results (a single value or multiple contributor values).
4. CloudWatch evaluates the aggregated results against your threshold using M-out-of-N evaluation on recent query executions.
5. If the threshold is breached, the alarm transitions to `ALARM` state and runs your configured actions (such as Amazon SNS notifications).

> **Note**
>
> Log Alarms evaluate the last N query executions. The alarm transitions to `ALARM` when M of those N executions breach the threshold.

To create a Log Alarm, see Create a Log Alarm.

Managed Scheduled Query lifecycle

When you create a Log Alarm, CloudWatch automatically creates an AWS managed Scheduled Query that runs your query on the specified schedule. You do not need to create the Scheduled Query separately.

The AWS managed Scheduled Query has the following characteristics:

- It is visible in the CloudWatch Logs console under Scheduled Queries.
- You cannot modify it directly. To change the query or its configuration, update the Log Alarm.
- CloudWatch deletes the AWS managed Scheduled Query when you delete the alarm.

Log Alarm configuration

A Log Alarm is configured with the following parameters:

- **QueryString** is the CloudWatch Logs Insights query to run.
- **LogGroupIdentifiers** are the log groups to query. Specify either log group names or log group ARNs.
- **ScheduledQueryRoleARN** is the ARN of the IAM role that allows CloudWatch Logs to run the scheduled query on your behalf.
- **AggregationExpression** defines how query results are aggregated into a numeric value for threshold evaluation.

- **ScheduleExpression** defines how frequently the query runs (for example, `rate(5 minutes)`).
- **StartTimeOffset** defines the lookback window in seconds for each query execution.
- **EndTimeOffset** defines the end of the query time range as an offset in seconds from the current time.
- **ComparisonOperator** is how aggregated results are compared to the threshold. Valid values: `GreaterThanThreshold`, `GreaterThanOrEqualToThreshold`, `LessThanThreshold`, `LessThanOrEqualToThreshold`.
- **Threshold** is the numeric value to compare against.
- **QueryResultsToEvaluate** is the number of recent query executions to evaluate (N in M-out-of-N).
- **QueryResultsToAlarm** is the number of breaching results required to trigger `ALARM` (M in M-out-of-N).
- **TreatMissingData** defines how missing query results are treated during evaluation.

For the full list of parameters and creation instructions, see Create a Log Alarm.

Logs query

The Log Alarm query is a CloudWatch Logs Insights query that selects and filters the log data to evaluate. The query runs on the log groups specified in `LogGroupIdentifiers` over the time range defined by `StartTimeOffset` and `EndTimeOffset`.

The query uses CloudWatch Logs Insights query syntax. For guidelines on writing efficient queries for Log Alarms, see Best practices and troubleshooting.

Aggregation expressions

The aggregation expression defines how CloudWatch summarizes query results into a numeric value for threshold evaluation. The expression uses the same syntax as the `stats` command in CloudWatch Logs Insights.

The syntax for an aggregation expression is as follows:

```
statistic_func_expression [by field1, field2, ...] [| sort asc|desc]
```

You can specify only a single aggregation expression. The following table lists the supported aggregation functions.

Function	Description	Example
`count(*)`	Count of all matched log lines.	`count(*)`
`avg(field)`	Average value of the specified field.	`avg(duration)`
`sum(field)`	Sum of the specified field.	`sum(bytesSent)`
`min(field)`	Minimum value of the specified field.	`min(latency)`
`max(field)`	Maximum value of the specified field.	`max(latency)`

The `bin()` function is not supported in the aggregation expression `by` clause. However, you can use `bin()` in the query string itself.

Multi-contributor alarms

When you include a `by` clause in your aggregation expression, the alarm evaluates each unique combination of field values (called a *contributor*) independently. The alarm transitions to `ALARM` state if any contributor breaches the threshold.

For example, the following expression groups error counts by service name:

```
count(*) by serviceName
```

Each unique value of `serviceName` is evaluated independently against the threshold. If any service exceeds the threshold in M out of N query executions, the alarm enters `ALARM` state.

The following limits apply to multi-contributor alarms:

- Maximum 5 fields in the `by` clause.
- Maximum 500 contributor results returned per query execution.
- Maximum 100 contributors tracked in `ALARM` state simultaneously.

By default, contributors are sorted alphabetically and only the first 500 are returned per query execution. To sort contributors by their aggregated value instead, specify `| sort asc` or `| sort`

`desc` in your aggregation expression (for example, `avg(latency) by serviceName | sort desc`). Value-based sorting ensures that the most significant contributors are evaluated first when the total number exceeds 500.

For multi-contributor alarms, Amazon SNS and Lambda actions run at the contributor level (once per breaching contributor). Systems Manager OpsItem actions run at the alarm level.

> **Note**
>
> Systems Manager Incident Manager and investigation actions are not supported for Log Alarms.

If a contributor disappears from query results (for example, an ephemeral resource is terminated), that contributor transitions to OK state regardless of the missing data treatment setting.

Missing data treatment

Missing data occurs when a scheduled query execution does not produce a value that can be evaluated against the threshold. This happens in the following cases:

No logs present — The log group contains no log events in the query time range.

Query returns no applicable results — Logs are present but the aggregation expression cannot produce a value. This happens when:

- Matching query results were not present as per the query filter.
- The field referenced in the aggregation expression was not present in the query results. For example, `count(error-codes)` where `error-codes` does not exist in the returned log events.

Note that `count(*)` on an empty result set returns 0, which is a valid datapoint and is not treated as missing.

You can configure how the alarm treats missing data using the `TreatMissingData` parameter. The following table describes the available options.

Value	Behavior
`missing`	Treat the datapoint as missing. This is the default.

Value	Behavior
notBreaching	Treat the missing datapoint as not breaching the threshold.
breaching	Treat the missing datapoint as breaching the threshold.
ignore	Ignore the missing datapoint and evaluate only available data.

Evaluation states

In addition to the standard OK, ALARM, and INSUFFICIENT_DATA states, Log Alarms can report the following evaluation states in the EvaluationState field. These states provide additional context about why the alarm is in its current state.

State	Description
EVALUATION_FAILURE	A transient CloudWatch service issue prevented evaluation. This can occur when the service experiences issues in evaluating query results due to service errors, or when some (but not all) query results failed. The alarm transitions to INSUFFICIENT_DATA . We recommend manual monitoring until the issue is resolved.
EVALUATION_ERROR	A client configuration error prevented evaluation. This can occur due to insufficient permissions, an invalid query, or when all query results have failed. The alarm transitions to INSUFFICIENT_DATA immediately. Refer to the StateReason field for details.
PARTIAL_DATA	The query returned the maximum 500 contributor groups but more matched. The alarm evaluates the available contributors, but results might be incomplete.

Alarm update

When you update the query, aggregation expression, schedule, or log groups of a Log Alarm, the alarm transitions to `INSUFFICIENT_DATA` until sufficient new datapoints are collected. Changes to the threshold or M-out-of-N values do not trigger this reset.

Actions and notifications

Log Alarms support the following actions:

- Amazon SNS notifications
- Lambda function invocations
- Systems Manager OpsItem creation

For the full actions support matrix, see Alarm actions.

When a Log Alarm transitions state, the action notification includes the following information:

- Standard alarm configuration change information (alarm name, description, configuration details).
- State change information (new state, state reason, timestamp).
- Amazon SNS email notifications also include a deep link to the CloudWatch Logs Insights console showing the full query results.

The following example shows an Amazon SNS email notification for a single-value Log Alarm (without a BY clause):

```
{
    "AlarmName": "HighErrorCount",
    "NewStateValue": "ALARM",
    "NewStateReason": "Threshold Crossed: 3 out of the last 5 query results
 [142.0 (10/06/26 12:15:00), 135.0 (10/06/26 12:10:00), 120.0 (10/06/26 12:05:00)]
 were greater than the threshold (100.0) (minimum 3 datapoints for OK -> ALARM
 transition).",
    "NewStateReasonData": {
        "version": "1.0",
        "queryDate": "2026-06-10T12:15:30.000+0000",
        "threshold": 100.0,
        "queryResultsToEvaluate": 5,
```

```
        "queryResultsToAlarm": 3,
        "results": [
            {
                "queryResultId": "scheduled-query-execution-id-3",
                "status": "COMPLETE",
                "timestamp": "2026-06-10T12:15:00.000+0000",
                "value": 142.0
            }
            // Additional results...
        ]
    },
    "StateChangeTime": "2026-06-10T12:15:30.000+0000",
    "OldStateValue": "OK"
    // Additional fields...
}
```

The following example shows an Amazon SNS email notification for a multi-contributor Log Alarm (with a BY clause). Each breaching contributor generates a separate notification:

```
{
    "AlarmName": "EndpointLatency",
    "NewStateValue": "ALARM",
    "NewStateReason": "5 out of 10 contributors evaluated to ALARM",
    "StateChangeTime": "2026-06-10T12:20:15.000+0000",
    "OldStateValue": "OK",
    "AlarmContributorId": "a1b2c3d4e5f6g7h8",
    "AlarmContributorAttributes": {
        "endpoint": "/api/orders"
    }
    // Additional fields...
}
```

Including log lines in notifications

You can optionally include raw query result log lines in alarm notifications by setting the `ActionLogLineCount` parameter to a value between 1 and 50. These are the underlying log events on which the aggregation expression is evaluated, not the aggregated values. The default value is 0, which means no log lines are included.

> **Note**
>
> Log lines are included only in Amazon SNS email notifications. Lambda actions do not include log lines in their payloads. CloudWatch also does not include log lines for multi-contributor alarms, even when you set `ActionLogLineCount`. CloudWatch includes log lines only for single-value Log Alarms that do not use a by clause.

> **Important**
>
> Including log lines in notifications might expose sensitive data from your logs in Amazon SNS messages. Review your log content before you enable this feature.

To include log lines, the log lines role must have the `logs:GetQueryResults` permission. The number of log lines included in a notification is limited by the requested count, the total results available, and the Amazon SNS payload size limit.

Best practices and troubleshooting

Best practices

Query optimization

- Test queries manually in CloudWatch Logs Insights before using them in a Log Alarm to verify performance and expected results.
- Use filter commands early in your query to reduce the volume of data processed.
- Limit query time ranges (StartTimeOffset) to avoid timeouts with high-volume log groups.
- Use field indexes to optimize query performance.

Schedule planning

- Choose a schedule frequency that allows queries to complete before the next execution. For high-volume log groups, use longer intervals (for example, 10 minutes instead of 5).
- Account for log ingestion delays when setting StartTimeOffset. A small gap between EndTimeOffset and the current time helps avoid evaluating incomplete data.

- Spread out Log Alarm schedules across your account to avoid hitting Scheduled Query concurrency limits. Concurrent query executions across your account cannot exceed 100. Factor in this quota when creating multiple Log Alarms with overlapping schedules.

Threshold tuning

- Start with higher QueryResultsToEvaluate (N) values to reduce alarm noise from transient spikes.
- For sparse events (such as errors that rarely occur), set TreatMissingData to `notBreaching` to keep the alarm in OK state when no logs match.
- For continuous signals (such as traffic logs), consider setting TreatMissingData to `breaching` to detect when expected log data stops arriving.

Multi-contributor design

- Choose meaningful fields for the BY clause that represent distinct resources or dimensions you want to monitor independently.
- Be aware that only the first 500 contributors are returned per query execution. If you expect more, narrow your query or use fewer BY clause fields.
- Use the `| sort desc` or `| sort asc` suffix in your aggregation expression to prioritize the highest or lowest values based on your comparison operator when the 500 contributor limit is reached.

Troubleshooting

Alarm stays in INSUFFICIENT_DATA

Possible cause	Resolution
Scheduled query execution role lacks permissions	Verify the role has `logs:StartQuery` , `logs:StopQuery` , `logs:GetQueryResults` , and `logs:DescribeLogGroups` permissions scoped to the correct log groups.
Log group does not exist or was deleted	Verify the log group ARNs in the alarm configuration are correct and accessible.

Possible cause	Resolution
Recently created or updated alarm	After creation or configuration update, the alarm remains in INSUFFICIENT_DATA until enough query executions complete to satisfy the M-out-of-N evaluation window.
Scheduled query is not running	Check the AWS managed Scheduled Query in the CloudWatch Logs console to verify it is executing on schedule.
Aggregation field not present in query results	The field referenced in the aggregation expression must be present in the query results. For example, if your aggregation is `avg(latency)` , make sure the query produces a `latency` field. If the field is not present, the result is treated as missing data.
Log ingestion delay	A scheduled query can only evaluate log events that have been ingested by the time it runs. `StartTimeOffset` and `EndTimeOffset` define the query window relative to execution time T — [T − StartTimeOffset, T − EndTimeOffset] — but they do not account for ingestion delay. If events are still being ingested for the window you query, the query runs before they are available and skips them. Use `EndTimeOffset` to shift the window back far enough that ingestion is complete for the entire range. Example: Suppose logs take up to 2 minutes to become queryable after the events occur. • `StartTimeOffset=60, EndTimeOffset=0` — window [T−60s, T]. The window ends at execution time, so recent events are not yet ingested and are missed. • `StartTimeOffset=180, EndTimeOffset=120` — window [T−180s, T−120s]. The window ends 2 minutes in the past, by which point all events are ingested and evaluable.

Alarm shows EVALUATION_ERROR

This indicates a client configuration issue. Check the StateReason field for details. Common causes:

- Invalid or malformed query syntax.
- Insufficient permissions on the scheduled query execution role.
- All query executions failed (for example, log group permissions revoked).

Alarm shows EVALUATION_FAILURE

This indicates a transient CloudWatch service issue. The alarm automatically recovers when the issue resolves. If it persists beyond a few minutes, check the CloudWatch service health dashboard.

Alarm shows PARTIAL_DATA

The query returned the maximum 500 contributor groups but more matched. The alarm evaluates available contributors, but results might be incomplete. Consider narrowing your query or reducing the number of BY clause fields.

Log lines not appearing in notifications

- Verify `ActionLogLineCount` is set to a value between 1 and 50.
- Verify the log lines role has `logs:GetQueryResults` permission scoped to the correct log groups.
- Log lines are included only in Amazon SNS email notifications. Other action types do not include log lines.
- Queries using `unmask()` cannot include log lines in notifications (rejected at creation time).

For additional best practices on query optimization, monitoring, and authorization, see Scheduled Queries best practices in the *Amazon CloudWatch Logs User Guide*.

Composite alarms

With CloudWatch, you can combine several alarms into one *composite alarm* to create a summarized, aggregated health indicator over a whole application or group of resources. Composite alarms are alarms that determine their state by monitoring the states of other alarms. You define rules to combine the status of those monitored alarms using Boolean logic.

You can use composite alarms to reduce alarm noise by taking actions only at an aggregated level. For example, you can create a composite alarm to send a notification to your web server team if any alarm related to your web server triggers. When any of those alarms goes into the ALARM state, the composite alarm goes itself in the ALARM state and sends a notification to your team. If other alarms related to your web server also go into the ALARM state, your team does not get overloaded with new notifications since the composite alarm has already notified them about the existing situation.

You can also use composite alarms to create complex alarming conditions and take actions only when many different conditions are met. For example, you can create a composite alarm that combines a CPU alarm and a memory alarm, and would only notify your team if both the CPU and the memory alarms have triggered.

Using composite alarms

When you use composite alarms, you have two options:

- Configure the actions you want to take only at the composite alarm level, and create the underlying monitored alarms without actions
- Configure a different set of actions at the composite alarm level. For example, the composite alarm actions could engage a different team in case of a widespread issue.

Composite alarms can take only the following actions:

- Notify Amazon SNS topics
- Invoke Lambda functions
- Create OpsItems in Systems Manager Ops Center
- Create incidents in Systems Manager Incident Manager

> **Note**
>
> All the underlying alarms in your composite alarm must be in the same account and the same Region as your composite alarm. However, if you set up a composite alarm in a CloudWatch cross-account observability monitoring account, the underlying alarms can watch metrics in different source accounts and in the monitoring account itself. For more information, see CloudWatch cross-account observability.

A single composite alarm can monitor 100 underlying alarms, and 150 composite alarms can monitor a single underlying alarm.

Rule expressions

All composite alarms contain rule expressions. Rule expressions tell composite alarms which other alarms to monitor and determine their states from. Rule expressions can refer to metric alarms and composite alarms. When you reference an alarm in a rule expression, you designate a function to the alarm that determines which of the following three states the alarm will be in:

- ALARM

  ALARM ("alarm-name or alarm-ARN") is TRUE if the alarm is in ALARM state.
- OK

  OK ("alarm-name or alarm-ARN") is TRUE if the alarm is in OK state.
- INSUFFICIENT_DATA

  INSUFFICIENT_DATA ("alarm-name or alarm-ARN") is TRUE if the named alarm is in INSUFFICIENT_DATA state.

Note

TRUE always evaluates to TRUE, and FALSE always evaluates to FALSE.

Alarm references

When referencing an alarm, using either the alarm name or ARN, the rule syntax can support referencing the alarm with or without quotation marks (") around the alarm name or ARN.

- If specified without quotes, alarm names or ARNs must not contain spaces, round brackets, or commas.
- If specified within quotes, alarm names or ARNs that *include* double quotes (") must enclose the " using backslash escape (\) characters for correct interpretation of the reference.

Syntax

The syntax of the expression you use to combine several alarms into one composite alarm uses boolean logic and functions. The following table describes the operators and functions available in rule expressions:

Operator/Function	Description
`AND`	Logical AND operator. Returns TRUE when all specified conditions are TRUE.
`OR`	Logical OR operator. Returns TRUE when at least one of the specified conditions is TRUE.
`NOT`	Logical NOT operator. Returns TRUE when the specified condition is FALSE.
`AT_LEAST`	Function that returns TRUE when a minimum number or percentag e of specified alarms are in the required state. Format: `AT_LEAST( M, STATE_CONDITION, (alarm1, alarm2, ...alarmN))` where M can be an absolute number or percentage (for example, 50%), and STATE_CONDITION can be ALARM, OK, INSUFFICIENT_DATA, NOT ALARM, NOT OK, or NOT INSUFFICIENT_DATA.

You can use parentheses to group conditions and control the order of evaluation in complex expressions.

Example expressions

The request parameter `AlarmRule` supports the use of the logical operators `AND`, `OR`, and `NOT`, as well as the `AT_LEAST` function, so you can combine multiple functions into a single expressions. The following example expressions show how you can configure the underlying alarms in your composite alarm:

- `ALARM(CPUUtilizationTooHigh) AND ALARM(DiskReadOpsTooHigh)`

  The expression specifies that the composite alarm goes into `ALARM` only if `CPUUtilizationTooHigh` and `DiskReadOpsTooHigh` are in `ALARM`.
- `AT_LEAST(2, ALARM, (WebServer1CPU, WebServer2CPU, WebServer3CPU, WebServer4CPU))`

The expression specifies that the composite alarm goes into ALARM when at least 2 out of the 4 web server CPU alarms are in ALARM state. This allows you to trigger alerts based on a threshold of affected resources rather than requiring all or just one to be in alarm state.

- `AT_LEAST(50%, OK, (DatabaseConnection1, DatabaseConnection2, DatabaseConnection3, DatabaseConnection4))`

  The expression specifies that the composite alarm goes into ALARM when at least 50% of the database connection alarms are in OK state. Using percentages allows the rule to adapt dynamically as you add or remove monitored alarms.

- `ALARM(CPUUtilizationTooHigh) AND NOT ALARM(DeploymentInProgress)`

  The expression specifies that the composite alarm goes into ALARM if CPUUtilizationTooHigh is in ALARM and DeploymentInProgress is not in ALARM. This is an example of a composite alarm that reduces alarm noise during a deployment window.

- `AT_LEAST(2, ALARM, (AZ1Health, AZ2Health, AZ3Health)) AND NOT ALARM(MaintenanceWindow)`

  The expression specifies that the composite alarm goes into ALARM when at least 2 out of 3 availability zone health alarms are in ALARM state and the maintenance window alarm is not in ALARM. This combines the AT_LEAST function with other logical operators for more complex monitoring scenarios.

Alarm suppression

Composite alarm action suppression allows you to temporarily disable alarm actions without deleting or modifying the alarm configuration. This is useful during planned maintenance, deployments, or when investigating known issues.

With composite alarm action suppression, you define alarms as suppressor alarms. Suppressor alarms prevent composite alarms from taking actions. For example, you can specify a suppressor alarm that represents the status of a supporting resource. If the supporting resource is down, the suppressor alarm prevents the composite alarm from sending notifications.

When to use alarm suppression

Common situations where alarm suppression is useful:

- Maintenance windows of your application

- Application deployments
- Ongoing incident investigation
- Testing and development activities

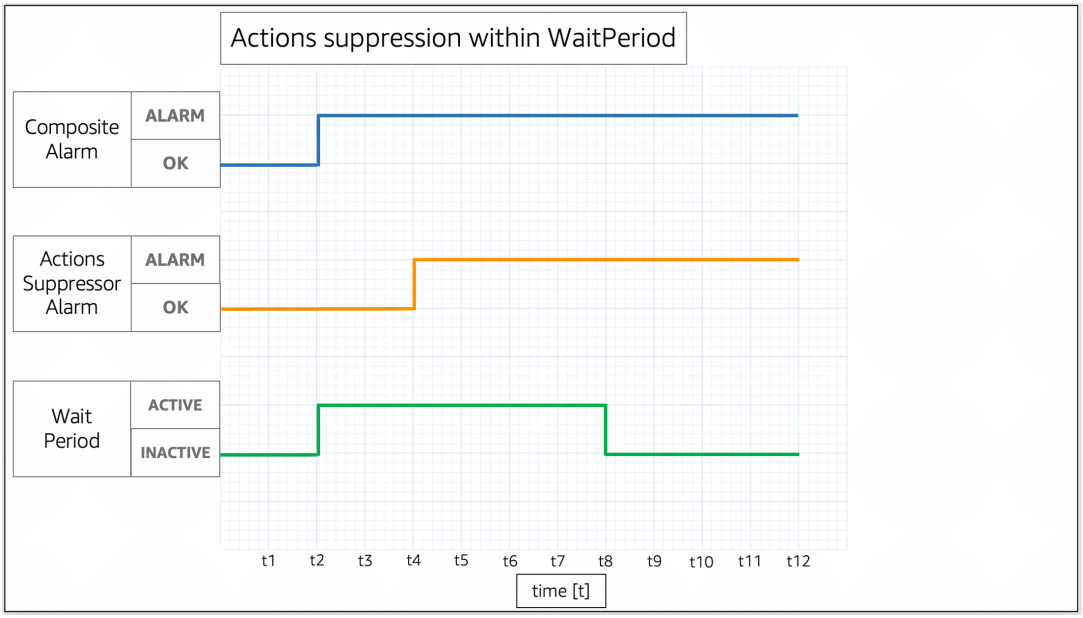
How suppressor alarms work

You specify suppressor alarms when you configure composite alarms. Any alarm can function as a suppressor alarm. When a suppressor alarm changes states from `OK` to `ALARM`, its composite alarm stops taking actions. When a suppressor alarm changes states from `ALARM` to `OK`, its composite alarm resumes taking actions.

Because composite alarms allow you to get an aggregated view of your health across multiple alarms, there are common situations where it is expected for those alarms to trigger. For example, during a maintenance window of your application or when you investigate an ongoing incident. In such situations, you may want to suppress the actions of your composite alarms, to prevent unwanted notifications or the creation of new incident tickets

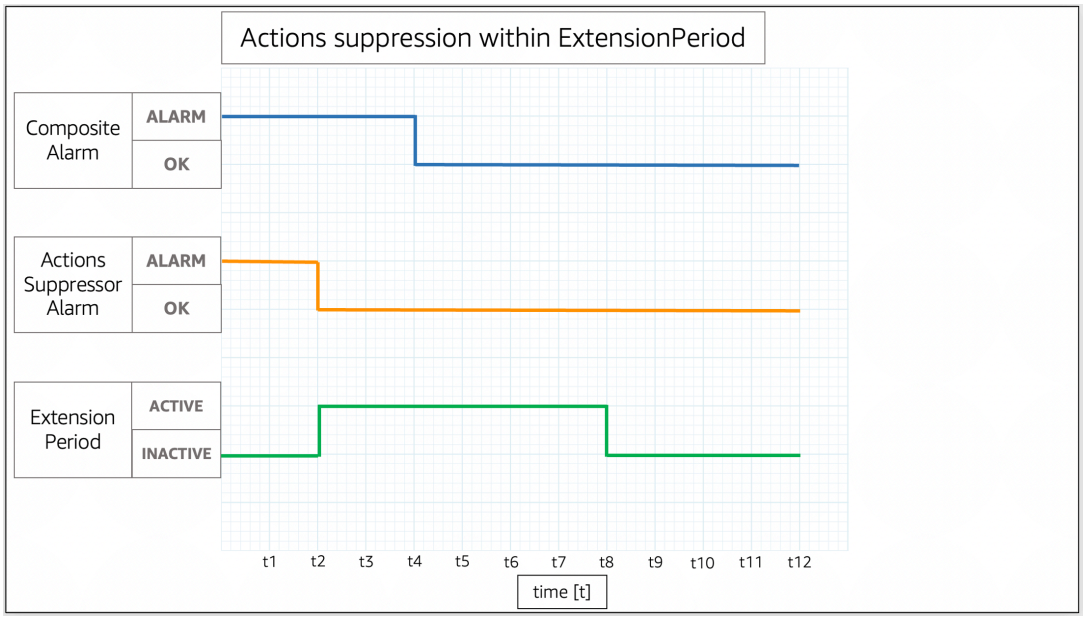
`WaitPeriod` and `ExtensionPeriod`

When you specify a suppressor alarm, you set the parameters `WaitPeriod` and `ExtensionPeriod`. These parameters prevent composite alarms from taking actions unexpectedly while suppressor alarms change states. Use `WaitPeriod` to compensate for any delays that can occur when a suppressor alarm changes from `OK` to `ALARM`. For example, if a suppressor alarm changes from `OK` to `ALARM` within 60 seconds, set `WaitPeriod` to 60 seconds.



In the image, the composite alarm changes from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t8. This gives the suppressor alarm time to change states from OK to ALARM at t4 before it suppresses the composite alarm's actions when the `WaitPeriod` expires at t8.

Use `ExtensionPeriod` to compensate for any delays that can occur when a composite alarm changes to OK following a suppressor alarm changing to OK. For example, if a composite alarm changes to OK within 60 seconds of a suppressor alarm changing to OK, set `ExtensionPeriod` to 60 seconds.



In the image, the suppressor alarm changes from ALARM to OK at t2. An `ExtensionPeriod` starts at t2 and ends at t8. This gives the composite alarm time to change from ALARM to OK before the `ExtensionPeriod` expires at t8.

Composite alarms don't take actions when `WaitPeriod` and `ExtensionPeriod` become active. Composite alarms take actions that are based on their currents states when `ExtensionPeriod` and `WaitPeriod` become inactive. We recommend that you set the value for each parameter to 60 seconds, as evaluates metric alarms every minute. You can set the parameters to any integer in seconds.

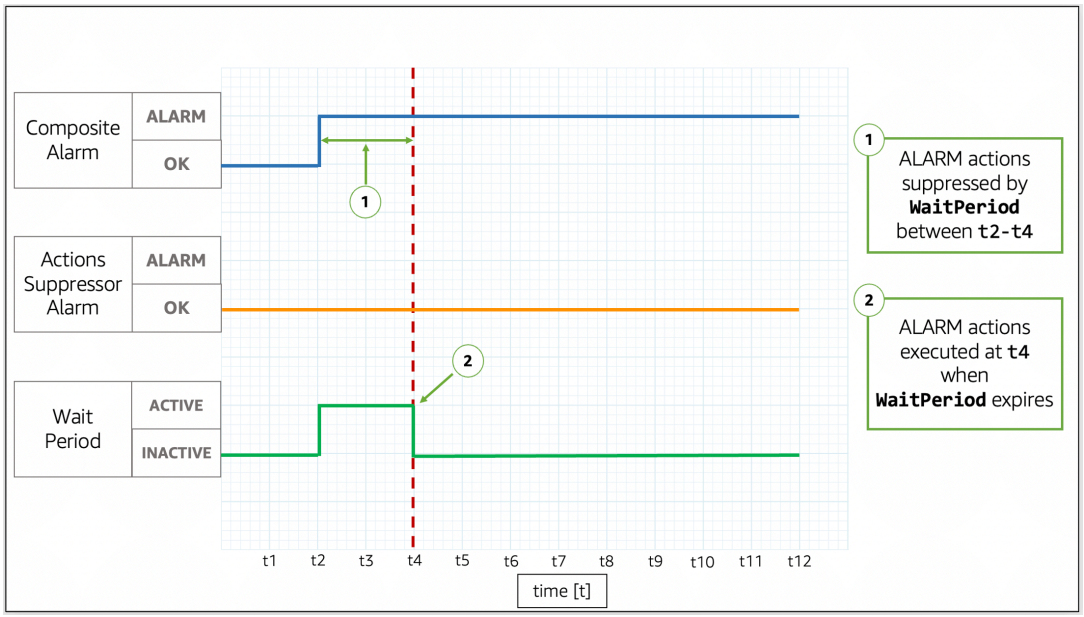
The following examples describe in more detail how `WaitPeriod` and `ExtensionPeriod` prevent composite alarms from taking actions unexpectedly.

Note

In the following examples, `WaitPeriod` is configured as 2 time units, and `ExtensionPeriod` is configured as 3 time units.

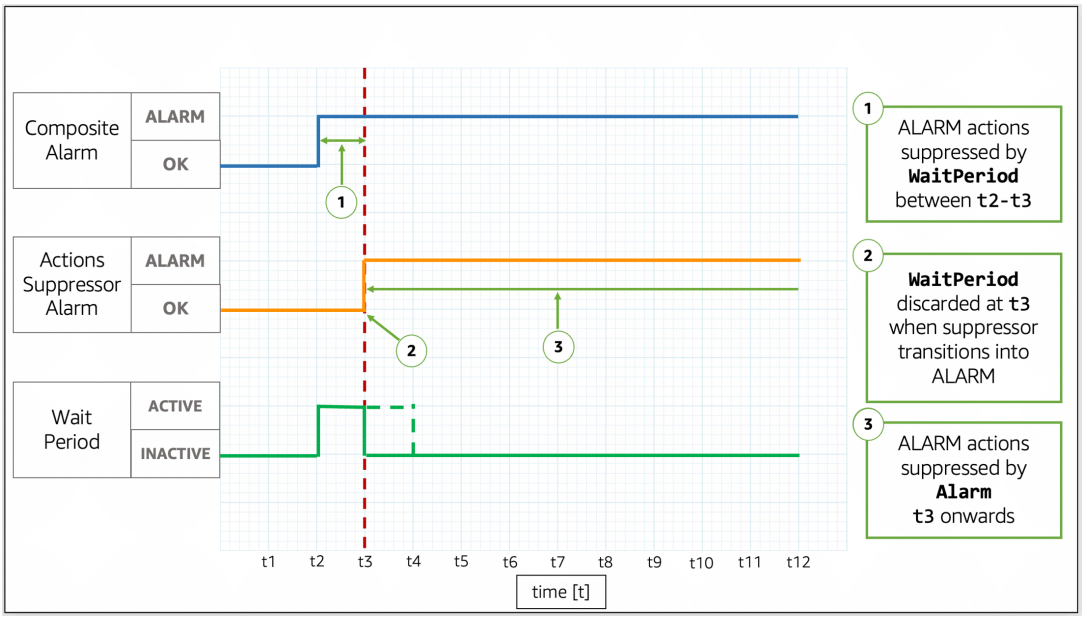
Examples

Example 1: Actions are not suppressed after `WaitPeriod`



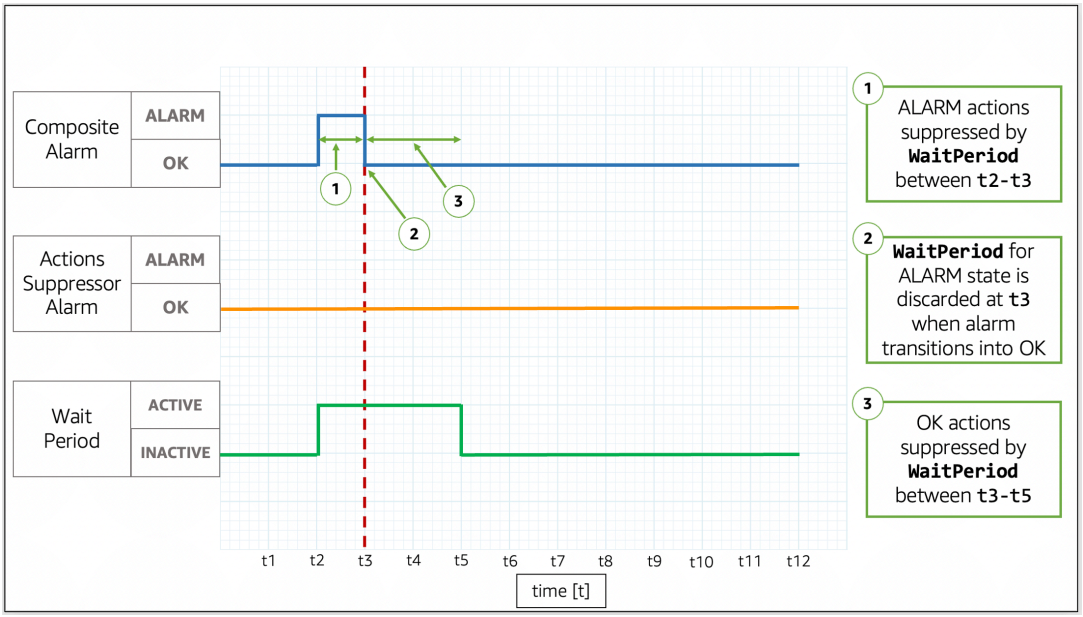
In the image, the composite alarm changes states from `OK` to `ALARM` at t2. A `WaitPeriod` starts at t2 and ends at t4, so it can prevent the composite alarm from taking actions. After the `WaitPeriod` expires at t4, the composite alarm takes its actions because the suppressor alarm is still in `OK`.

Example 2: Actions are suppressed by alarm before `WaitPeriod` expires



In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from OK to ALARM at t3. Because the suppressor alarm changes states from OK to ALARM at t3, the `WaitPeriod` that started at t2 is discarded, and the suppressor alarm now stops the composite alarm from taking actions.

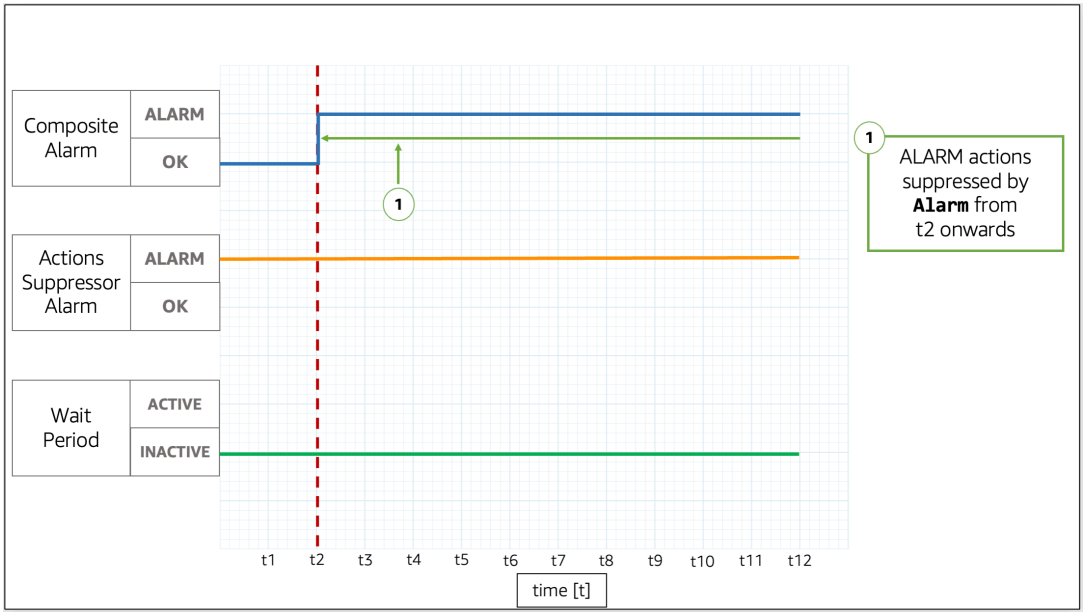
Example 3: State transition when actions are suppressed by `WaitPeriod`



In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states. The composite alarm

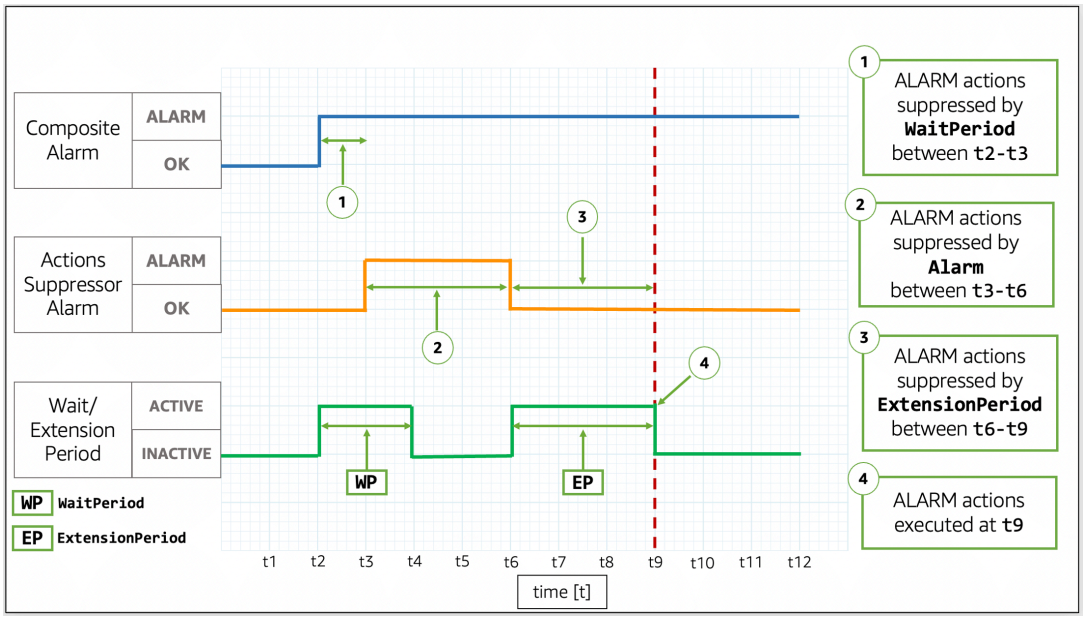
changes back to OK at t3, so the `WaitPeriod` that started at t2 is discarded. A new `WaitPeriod` starts at t3 and ends at t5. After the new `WaitPeriod` expires at t5, the composite alarm takes its actions.

Example 4: State transition when actions are suppressed by alarm



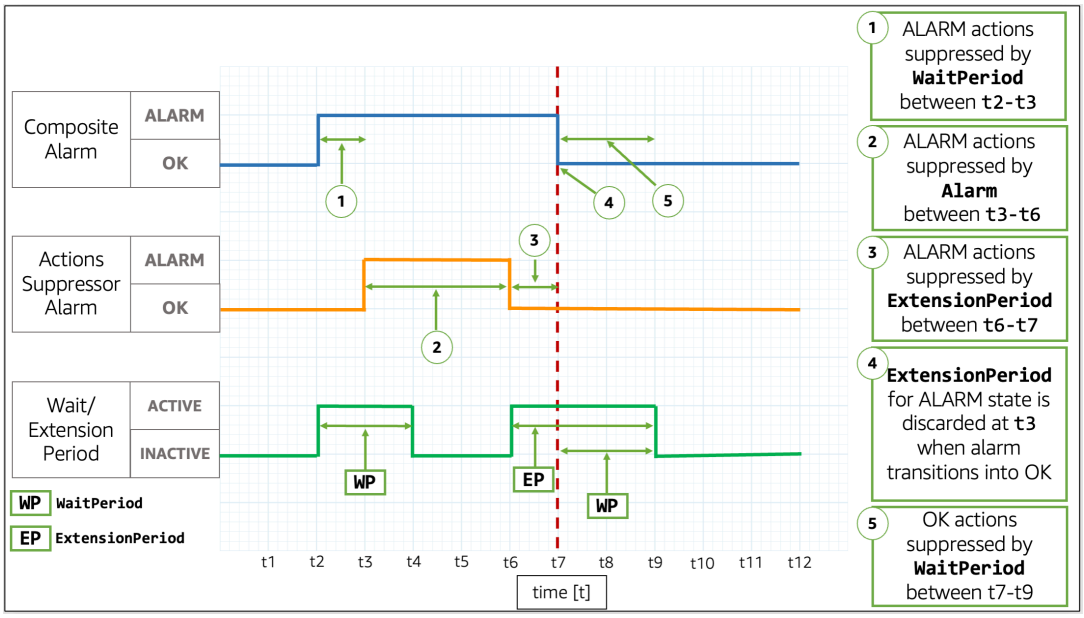
In the image, the composite alarm changes states from OK to ALARM at t2. The suppressor alarm is already in ALARM. The suppressor alarm stops the composite alarm from taking actions.

Example 5: Actions are not suppressed after `ExtensionPeriod`



In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from OK to ALARM at t3 before it suppresses the composite alarm's actions until t6. Because the suppressor alarm changes states from OK to ALARM at t3, the `WaitPeriod` that started at t2 is discarded. At t6, the suppressor alarm changes to OK. An `ExtensionPeriod` starts at t6 and ends at t9. After the `ExtensionPeriod` expires, the composite alarm takes its actions.

Example 6: State transition when actions are suppressed by `ExtensionPeriod`



In the image, the composite alarm changes states from OK to ALARM at t2. A `WaitPeriod` starts at t2 and ends at t4. This gives the suppressor alarm time to change states from OK to ALARM at t3 before it suppresses the composite alarm's actions until t6. Because the suppressor alarm changes states from OK to ALARM at t3, the `WaitPeriod` that started at t2 is discarded. At t6, the suppressor alarm changes back to OK. An `ExtensionPeriod` starts at t6 and ends at t9. When the composite alarm changes back to OK at t7, the `ExtensionPeriod` is discarded, and a new `WaitPeriod` starts at t7 and ends at t9.

> **Tip**
>
> If you replace the action suppressor alarm, any active `WaitPeriod` or `ExtensionPeriod` is discarded.

Action Suppression and Mute Rules

When both action suppression and alarm mute rules are active for a composite alarm, mute rules take precedence and suppress all alarm actions. After the mute window ends, the composite alarm's action suppression configuration determines whether actions are executed based on the suppressor alarm state and configured wait or extension periods. For more information about alarm mute rules, see the section called “Alarm Mute Rules”.

Alarm actions

You can specify what actions an alarm takes when it changes state between the OK, ALARM, and INSUFFICIENT_DATA states.

Most actions can be set for the transition into each of the 3 states. Except for Auto Scaling actions, the actions happen only on state transitions, and are not performed again if the condition persists for hours or days.

The following are supported as alarm actions:

- Notify one or more subscribers by using an Amazon Simple Notification Service topic. Subscribers can be applications as well as persons.
- Invoke a Lambda function. This is the easiest way for you to automate custom actions on alarm state changes.
- Alarms based on EC2 metrics can also perform EC2 actions, such as stopping, terminating, rebooting, or recovering an EC2 instance.
- Alarms can scale an Auto Scaling group or an Application Auto Scaling target. For more information, see Auto Scaling actions.
- Alarms can create OpsItems in Systems Manager Ops Center or create incidents in AWS Systems Manager Incident Manager. These actions are performed only when the alarm goes into ALARM state.
- An alarm can start an investigation when it goes into ALARM state.

Alarms also emit events to Amazon EventBridge when they change state, and you can set up Amazon EventBridge to trigger other actions for these state changes.

Alarm actions and notifications

The following table shows the actions executed for alarms along with their behavior for multiple time series (or contributors) alarms:

Action Type	Standard metric alarm	Metrics Insights Multiple Time Series Alarm	PromQL Alarm	Log Alarm	More Information
SNS notifications	Supported	Contributor Level	Contributor Level	Contributor Level	Amazon SNS event destinations
EC2 actions (stop, terminate, reboot, recover)	Supported	Not supported	Not supported	Not supported	Stop, terminate, reboot, or recover an EC2 instance
Auto Scaling group actions	Simple, step, and target tracking	Not supported	Not supported	Not supported	Auto Scaling actions
Application Auto Scaling actions	Step and target tracking	Not supported	Step	Not supported	Auto Scaling actions
Systems Manager OpsItem creation	Supported	Alarm Level	Not supported	Alarm Level	Configure CloudWatch alarms to create OpsItems
Systems Manager Incident Manager incidents	Supported	Alarm Level	Not supported	Not supported	Creating incidents automatically with CloudWatch alarms

Action Type	Standard metric alarm	Metrics Insights Multiple Time Series Alarm	PromQL Alarm	Log Alarm	More Information
Lambda function invocation	Supported	Contributor Level	Contributor Level	Contributor Level	Invoke a Lambda function from an alarm
CloudWatch investigations investigation	Supported	Alarm Level	Not supported	Not supported	Start a CloudWatch investigations from an alarm

The content of alarm notifications differs depending on the alarm type:

- Single-metric alarms include both a state reason and detailed state reason data, showing the specific datapoints that caused the state change.
- Multi-time series Metrics Insights alarms provide a simplified state reason for each contributor, without the detailed state reason data block.
- PromQL alarms do not include a state reason or state reason data in their notifications.
- Log alarms without contributors (single-value) include both a state reason and detailed state reason data, showing the query results evaluated.
- Log alarms with contributors (using a BY clause) provide a simplified state reason for each contributor, without the detailed state reason data block.

Example Notification Content Examples

Single-metric alarm notification includes detailed data:

```
{
  "stateReason": "Threshold Crossed: 3 out of the last 3 datapoints [32.6 (03/07/25
 08:29:00), 33.8 (03/07/25 08:24:00), 41.0 (03/07/25 08:19:00)] were greater than the
 threshold (31.0)...",
  "stateReasonData": {
    "version": "1.0",
```

```
    "queryDate": "2025-07-03T08:34:06.300+0000",
    "startDate": "2025-07-03T08:19:00.000+0000",
    "statistic": "Average",
    "period": 300,
    "recentDatapoints": [41, 33.8, 32.6],
    "threshold": 31,
    "evaluatedDatapoints": [
      {
        "timestamp": "2025-07-03T08:29:00.000+0000",
        "sampleCount": 5,
        "value": 32.6
      }
      // Additional datapoints...
    ]
  }
}
```

Multiple time series Metrics Insights Alarm SNS notification for Contributor example:

```
{
  "AlarmName": "DynamoDBInsightsAlarm",
  "NewStateValue": "ALARM",
  "NewStateReason": "Threshold Crossed: 1 datapoint was less than the threshold (1.0).
 The most recent datapoint which crossed the threshold: [0.0 (01/12/25 13:34:00)].",
  "StateChangeTime": "2025-12-01T13:42:04.919+0000",
  "OldStateValue": "OK",
  "AlarmContributorId": "6d442278dba546f6",
  "AlarmContributorAttributes": {
    "TableName": "example-dynamodb-table-name"
  }
  // Additional information...
}
```

PromQL Alarm SNS notification for Contributor example:

```
{
  "AlarmName": "HighCPUUsageAlarm",
  "NewStateValue": "ALARM",
  "StateChangeTime": "2025-12-01T13:42:04.919+0000",
  "OldStateValue": "OK",
  "AlarmContributorId": "1d502278dcd546a1",
  "AlarmContributorAttributes": {
```

```
    "team": "example-team-name"
  }
  // Additional information...
}
```

Auto Scaling actions

A CloudWatch metric alarm can trigger an Auto Scaling scaling policy to add or remove capacity. Composite alarms can't trigger Auto Scaling actions.

Scaling policies belong to one of two services:

- Amazon EC2 Auto Scaling scales the EC2 instances in an Auto Scaling group.
- Application Auto Scaling scales other resources, such as Amazon ECS services, Amazon DynamoDB tables, Amazon Aurora replicas, and Spot Fleets.

How the alarm is created depends on the policy type:

- For simple scaling and step scaling, you create a CloudWatch metric alarm and attach it to the scaling policy.
- For target tracking, the Auto Scaling service creates and manages the alarms for you (a high alarm and a low alarm). Don't edit or delete them.

For more information, see Step and simple scaling policies for Amazon EC2 Auto Scaling. For Application Auto Scaling, see Step scaling policies and Target tracking scaling policies.

Alarm tags in action payloads

You can include a CloudWatch alarm's tags in the payloads it sends when the alarm changes state. Recipients can then route, filter, or add context by using tags such as team, environment, or cost center. This applies to Amazon SNS notifications, Lambda function invocations, and the alarm state change events that CloudWatch sends to EventBridge.

To enable this, see Alarm tags in action payloads.

Muting Alarm Actions

Alarm mute rules allow you to automatically mute alarm actions during predefined time windows, such as maintenance periods or operational events. CloudWatch continues monitoring alarm states

while preventing unwanted notifications. For more information, see the section called “Alarm Mute Rules”.

Mute rules versus disabling alarm actions

Alarm mute rules temporarily mute actions during scheduled time windows and automatically unmute when the window ends. In contrast, the `DisableAlarmActions` API permanently disables alarm actions until you manually call `EnableAlarmActions`. The `EnableAlarmActions` API does not unmute alarms that are muted by active mute rules.

Note

Muting an alarm does not stop CloudWatch from sending alarm events for alarm create, update, delete, and state changes to Amazon EventBridge.

Alarm Mute Rules

Alarm Mute Rules is a CloudWatch feature that provides you a mechanism to automatically mute alarm actions during predefined time windows. When you create a mute rule, you define specific time periods and target alarms whose actions will be muted. CloudWatch will continue monitoring and evaluating alarm states while preventing unwanted notifications or automated alarm actions during expected operational events.

Alarm Mute Rules help you manage critical operational scenarios where alarm actions would be unnecessary or disruptive. For example, during planned maintenance windows, you can prevent automated alarm actions while your systems are intentionally offline or experiencing expected issues, allowing you to perform maintenance without interruptions. For operations during non-business hours such as weekends or holidays, you can mute non-critical alarm actions when immediate response is not required, reducing alarm noise and unnecessary notifications to your operations team. In testing environments, mute rules allow you to temporarily mute alarm actions during scenarios such as load testing where high resource usage or error rates are expected and don't require immediate attention. When your team is actively troubleshooting issues, mute rules allow you to prevent duplicate alarm actions from being triggered, helping you focus on resolution without being distracted by redundant alarm notifications.

Defining alarm mute rules

Alarm mute rules can be defined using: **rules** and **targets**.

- **Rules** - define the time windows when alarm actions should be muted. Rules are composed of three attributes:
  - **Expression** – Defines when the mute period begins and how it repeats. You can use two types of expressions:
    - **Cron expressions** – Use standard cron syntax to create recurring mute windows. This approach is ideal for regular maintenance schedules, such as weekly system updates or daily backup operations. Cron expressions allow you to specify complex recurring patterns, including specific days of the week, months, or times.

      Syntax for cron expression

      ```
      ############## minute (0 - 59)
      # ############## hour (0 - 23)
      # # ############## day of the month (1 - 31)
      # # # ############## month (1 - 12) (or JAN-DEC)
      # # # # ############## day of the week (0 - 6) (0 or 7 is Sunday, or MON-SUN)
      # # # # #
      # # # # #
      * * * * *
      ```

      - The following characters are supported in each field:
        - `Minute`: `* , - /`
        - `Hour`: `* , - /`
        - `Day of month`: `* ? , - / L`
        - `Month`: `* , - /`
        - `Day of week`: `* ? , - L #`

        For example, `cron(0 9 * * SUN#4)` activates on the fourth Sunday of every month at 9:00 AM.
      - English names can be used for the `month` (JAN-DEC) and `day of week` (SUN-SAT) fields
    - **At expressions** – Use at expressions for one-time mute windows. This approach works well for planned operational events that occur once at a known time.

```
Syntax: `at(yyyy-MM-ddThh:mm)`
```

- **Duration** – Specifies how long the mute rule lasts once activated. Duration must be specified in ISO-8601 format with a minimum of 1 minute (PT1M) and maximum of 15 days (P15D).
- **Timezone** – Specifies the timezone in which the mute window will be applied according to the expressions, using standard timezone identifiers such as "America/Los_Angeles" or "Europe/London".

- **Targets** - specify the list of alarm names whose actions will be muted during the defined time windows. You can include both metric alarms and composite alarms in your target list.

You can optionally include start and end timestamps to provide additional boundaries for your mute windows. Start timestamps make sure that mute rules don't activate before a specific date and time, while end timestamps prevent rules from being applied beyond the specified date and time.

For more information about creating alarm mute rules programmatically, see PutAlarmMuteRule.

Note

- The targeted alarms must be present in the same AWS account and same AWS Region in which the mute rule is created.
- A single alarm mute rule can target up to 100 alarms by alarm names.

The CloudWatch console includes a dedicated "Alarm Mute Rules" tab that provides centralized management of all your mute rules within your AWS account. You can search for specific mute rules using the mute rule attributes such as rule name.

Mute Rule Status

Once created, an alarm mute rule can be in one of the below three statuses:

- **SCHEDULED** – The mute rule will become active at some time in the future according to the configured time window expression.

- **ACTIVE** – The mute rule is currently active as per the configured time window expression and actively muting targeted alarm actions.
- **EXPIRED** – The mute rule will not be SCHEDULED/ACTIVE anymore in the future. This occurs for one-time mute rules after the mute window has ended, or for recurring mute rules when an end timestamp is configured and that time has passed.

Effects of mute rules on alarms

During an active mute window, when a targeted alarm changes state and has actions configured, CloudWatch mutes those actions from executing. Mutes are applied only to alarm actions, meaning that alarms continue to be evaluated and state changes are visible in the CloudWatch console. However, configured actions such as Amazon Simple Notification Service notifications, Amazon Elastic Compute Cloud Auto Scaling actions, or Amazon EC2 actions are prevented from executing. CloudWatch continues to evaluate alarm states normally throughout the mute period, and you can view this information through alarm history.

When a mute window ends, CloudWatch compares the current state of each targeted alarm with the state that the alarm was in when the mute window started. If the two states are different, CloudWatch automatically executes the actions configured for the alarm's current state. This ensures that your alarm actions are executed for ongoing issues after the planned mute period ends, maintaining the integrity of your monitoring system.

If the alarm is in the same state as it was when the mute window started, no actions are executed when the window ends. This is true even if the alarm changed state during the window and then returned to its original state before the window ended. The state in which an individual action was muted is not used for this comparison. A state change that happens after the mute window has ended is not affected by the mute rule, and its actions are executed as usual.

> **Note**
>
> When you mute an alarm:
>
> - All the actions associated with the targeted alarms are muted
> - Actions associated with all alarm states (OK, ALARM, and INSUFFICIENT_DATA) are muted

Viewing and managing muted alarms

For information about viewing and managing muted alarms, see Viewing and managing muted alarms.

How alarm mute rules work

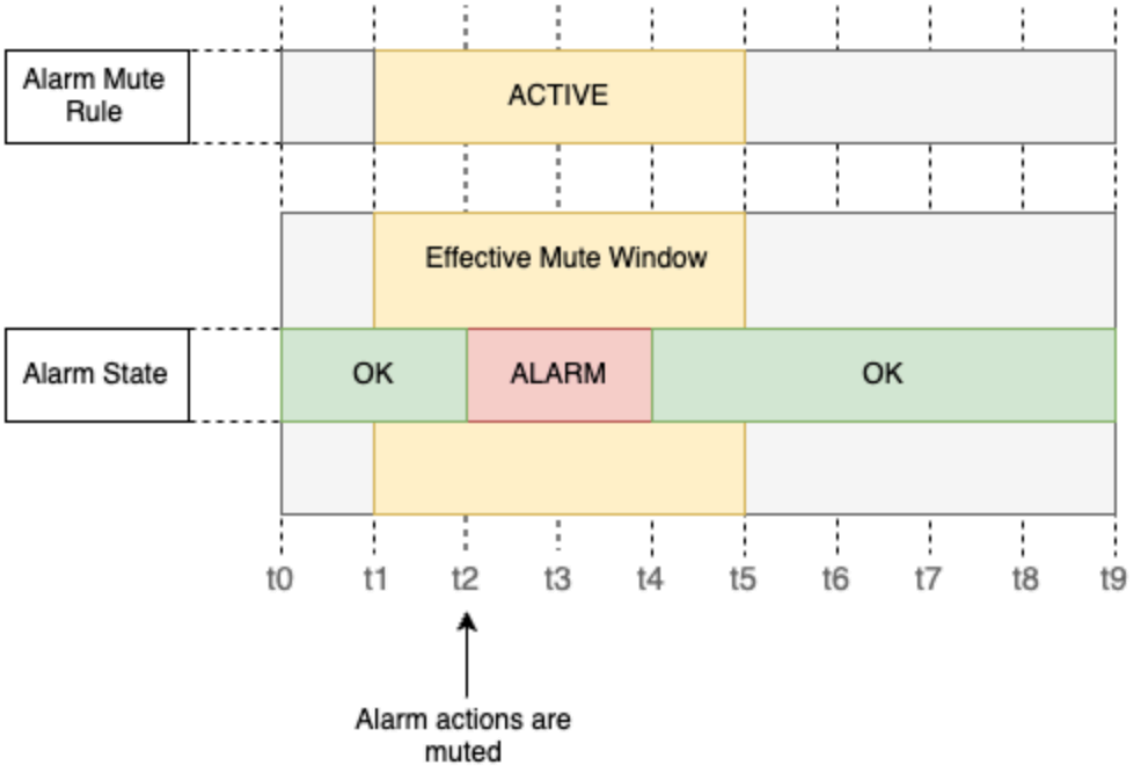
The following scenarios illustrate how alarm mute rules affect the targeted alarms and how the alarm actions are muted or executed.

> **Note**
>
> - Muting an alarm will mute actions associated with all alarm states, including OK, ALARM, and INSUFFICIENT_DATA. The behaviours illustrated below apply to actions associated with all alarm states.
> - When a mute window ends, CloudWatch compares the alarm's current state with the state the alarm was in when the mute window started. Actions are executed only if these two states are different, and the actions executed are those configured for the alarm's current state.
> - When you mute a Metrics Insights alarm, all contributor metric series for that alarm are automatically muted as well.

Scenario: Alarm actions are muted when a mute rule is active

Consider that,

- An Alarm has actions configured for its ALARM state
- An alarm mute rule is scheduled to be active from t1 to t5 that targets the Alarm

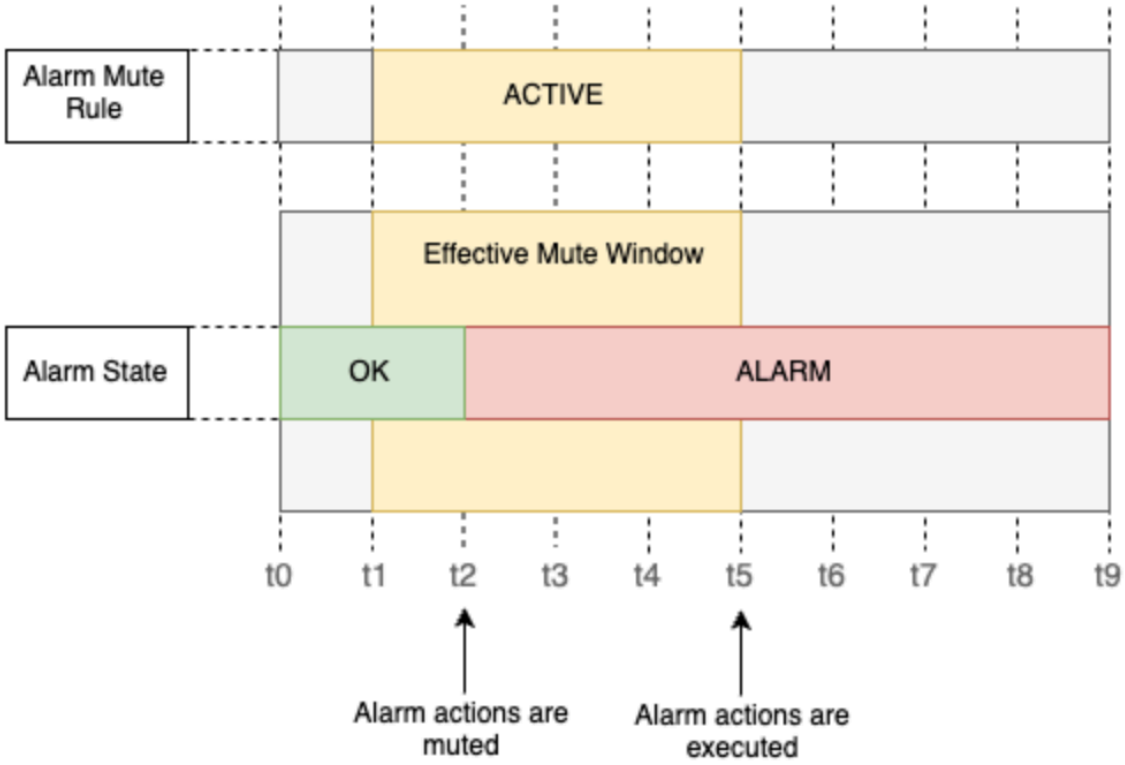


- At **t0** - Alarm is in OK state, mute rule status is SCHEDULED
- At **t1** - Mute rule status becomes ACTIVE
- At **t2** - Alarm transitions to ALARM state, action is muted as the alarm is effectively muted by the mute rule.
- At **t4** - Alarm returns to OK state while mute rule is still active
- At **t5** - Mute rule becomes inactive, but no ALARM action is executed because the alarm is in OK state, the same state it was in when the mute window started

Scenario: Alarm action muted when mute rule is active and re-triggered after mute window

Consider that,

- An Alarm has actions configured for its ALARM state
- An alarm mute rule is scheduled to be active from t1 to t5 that targets the Alarm



- At **t0** - Alarm is in OK state, mute rule status is SCHEDULED
- At **t1** - Mute rule status becomes ACTIVE
- At **t2** - Alarm transitions to ALARM state, action is muted as the alarm is effectively muted by the mute rule.
- At **t4** - Mute window becomes inactive, alarm is still in ALARM state
- At **t5** - Alarm action is executed because the mute window has ended and the alarm is in a different state (ALARM) from the state it was in when the mute window started (OK)

Scenario: Alarm is already in ALARM state when the mute window starts and is in ALARM state when it ends

Consider that,

- An Alarm has actions configured for its ALARM state and no actions configured for its OK state
- An alarm mute rule is scheduled to be active from t1 to t5 that targets the Alarm

- At **t0** - Alarm is in ALARM state, mute rule status is SCHEDULED. The ALARM action was executed when the alarm transitioned to ALARM state.

- At **t1** - Mute rule status becomes ACTIVE. The alarm is in ALARM state when the mute window starts.
- At **t2** - Alarm transitions to OK state. No OK action is configured, so there is nothing to mute.
- At **t3** - Alarm transitions back to ALARM state, the ALARM action is muted because the alarm is effectively muted by the mute rule.
- At **t5** - Mute rule becomes inactive. No ALARM action is executed, because the alarm is in the same state (ALARM) as it was when the mute window started. The fact that the ALARM action was muted at t3 does not cause it to be executed at the end of the window.

> **Note**
>
> If the alarm had instead been in OK state at t5, the OK actions (if any) would have been executed, because OK is a different state from the state the alarm was in when the mute window started.

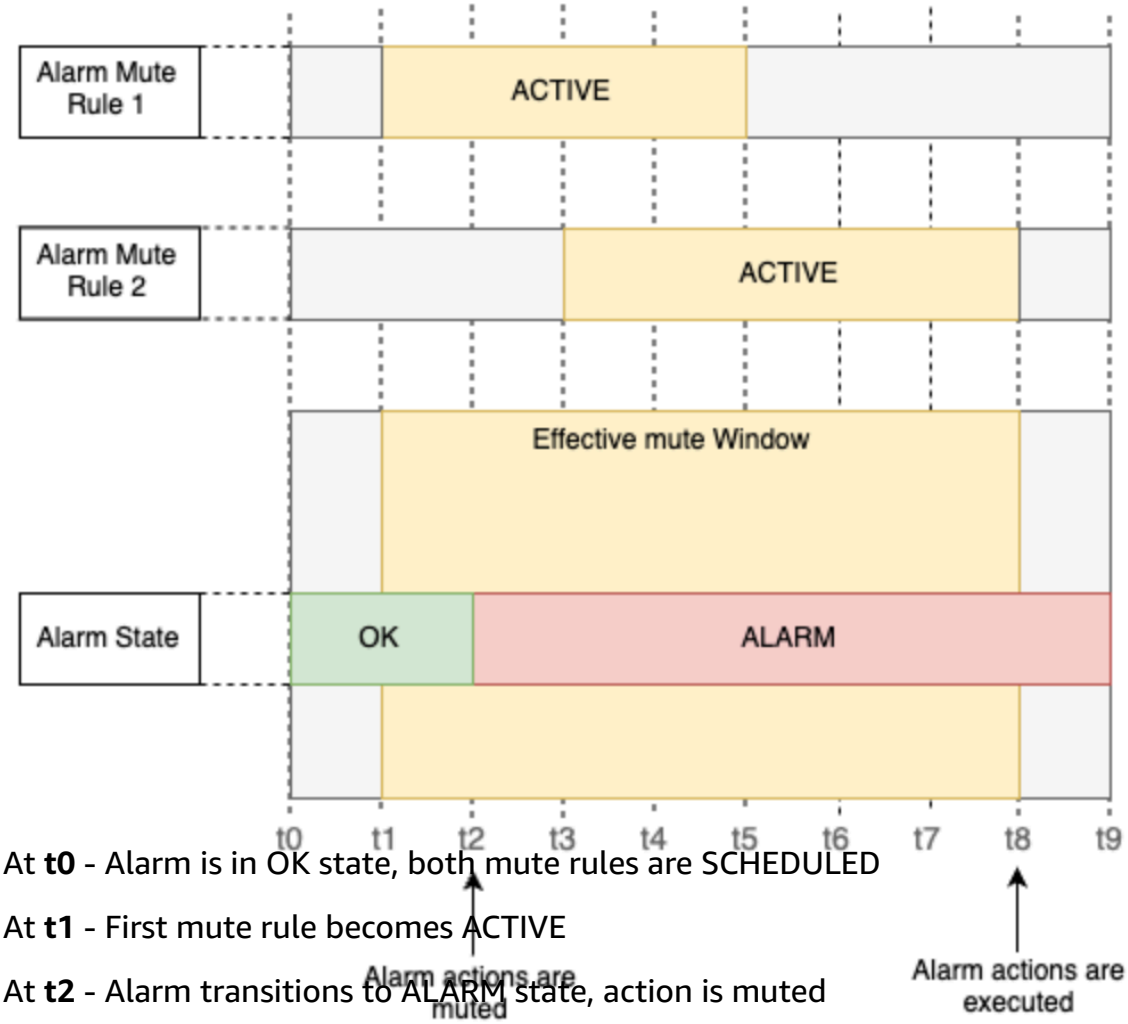
Scenario: Multiple overlapping alarm mute rules

Consider that,

- An Alarm has actions configured for its ALARM state

Consider that there are two mute rules,

- Alarm Mute Rule 1 - mutes Alarm from t1 to t5
- Alarm Mute Rule 2 - mutes Alarm from t3 to t9

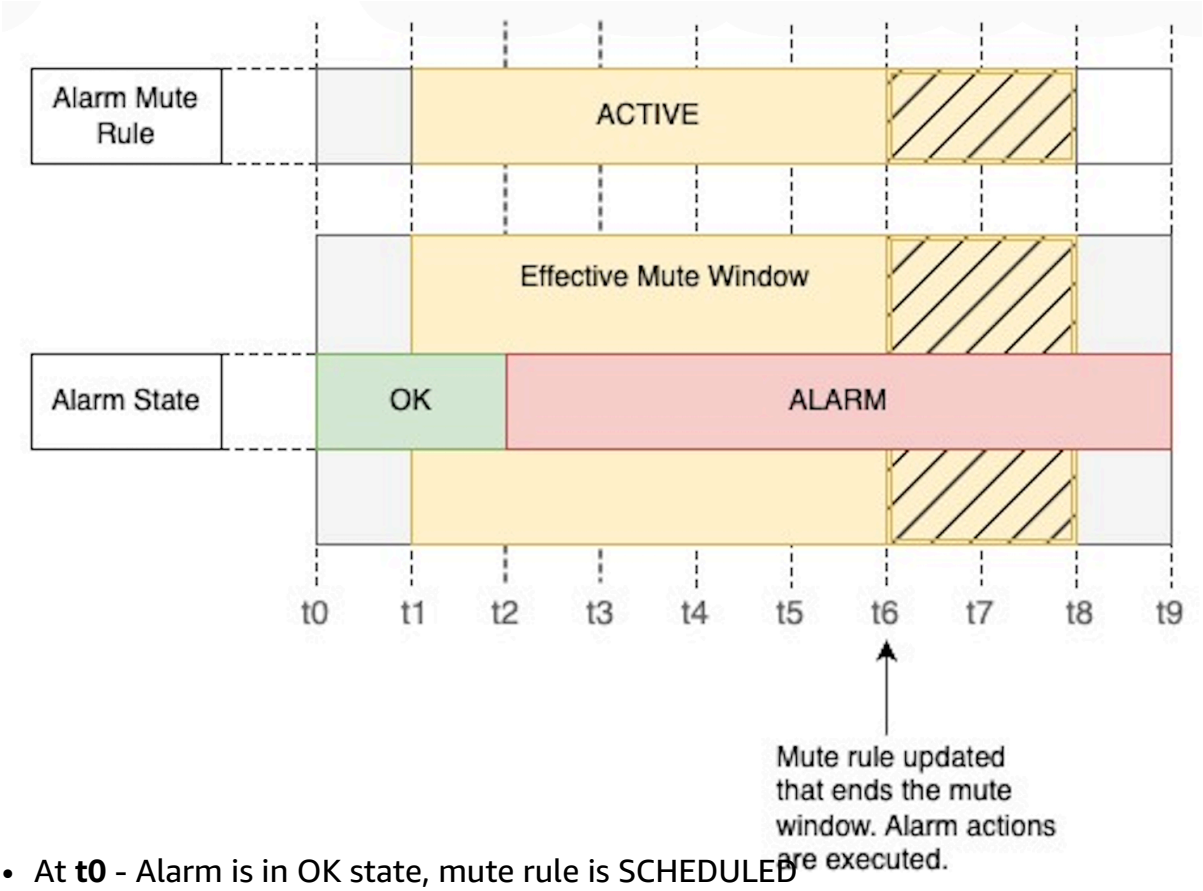


- At **t0** - Alarm is in OK state, both mute rules are SCHEDULED
- At **t1** - First mute rule becomes ACTIVE
- At **t2** - Alarm transitions to ALARM state, action is muted
- At **t3** - Second mute rule becomes ACTIVE
- At **t5** - First mute rule becomes inactive, but alarm action remains muted because second mute rule is still active
- At **t8** - Alarm action is executed because the second mute window has ended and the alarm is in a different state (ALARM) from the state it was in when the effective mute window started at t1 (OK)

Scenario: Muted alarm actions are executed when mute rule update ends the mute window

Consider that,

- An Alarm has actions configured for its ALARM state
- An alarm mute rule is scheduled to be active from t1 to t8 that targets the Alarm



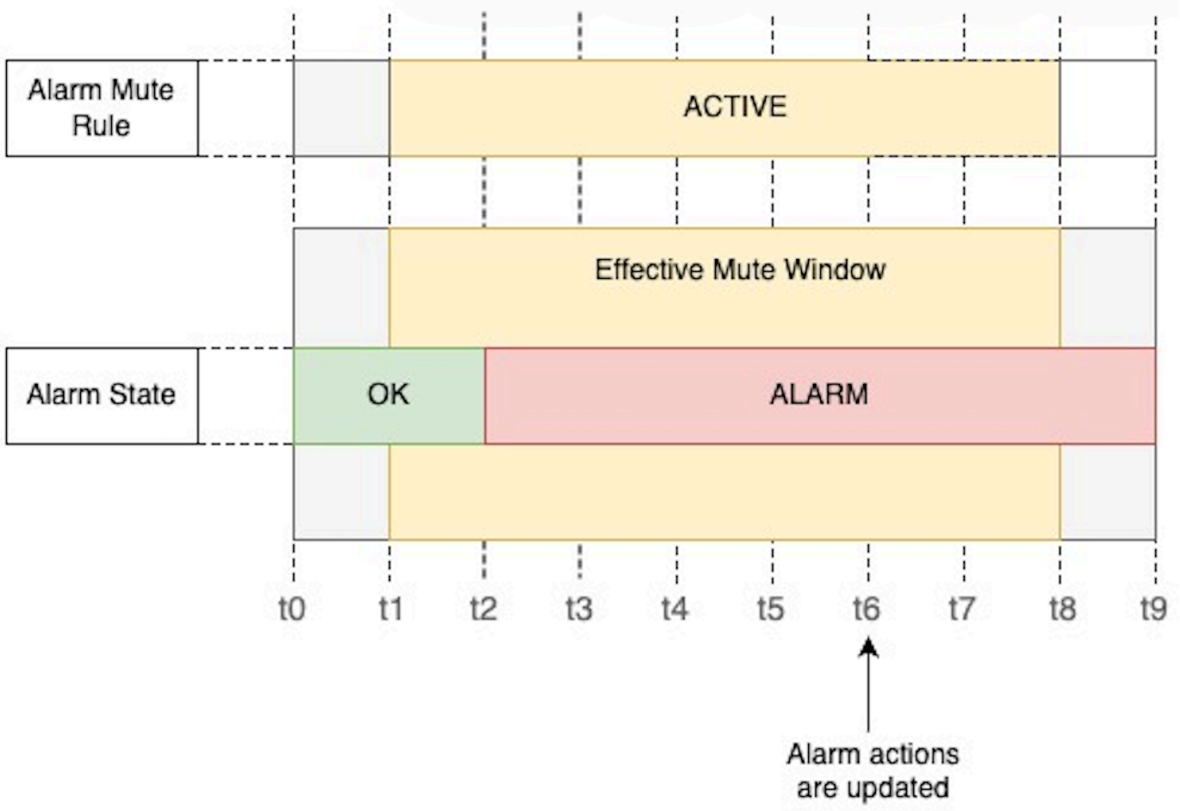
- At **t0** - Alarm is in OK state, mute rule is SCHEDULED
- At **t1** - Mute rule becomes ACTIVE
- At **t2** - Alarm transitions to ALARM state, actions are muted
- At **t6** - The mute rule configuration is updated such that the time window ends at t6. Alarm actions are immediately executed at t6 because the mute rule is no longer active.

> **Note**
>
> The same behaviour applies,
>
> - If the mute rule is deleted at t6. Deleting the mute rule immediately unmutes the alarm.
> - If the alarm is removed from the mute rule targets (at t6) then the alarm will be unmuted immediately.

Scenario: New actions are executed if alarm actions are updated during mute window

Consider that,

- An Alarm has actions configured for its ALARM state
- An alarm mute rule is scheduled to be active from t1 to t8 that targets the Alarm



- At **t0** - Alarm is in OK state, mute rule is SCHEDULED. An SNS action is configured with the alarm state ALARM.
- At **t1** - Mute rule becomes ACTIVE
- At **t2** - Alarm transitions to ALARM state, the configured SNS action is muted
- At **t6** - Alarm configuration is updated to remove SNS action and add Lambda action
- At **t8** - The lambda action configured to the Alarm is executed because the mute window has ended and the alarm is in a different state (ALARM) from the state it was in when the mute window started (OK). The actions executed are those configured for the alarm's current state at the end of the window.

> **Note**
>
> If all the alarm actions are removed during the mute window (at t6 in above example) then no actions will be executed at the end of mute window (at t8 above)

Example schedules for common use cases

The following examples show how to configure time window expressions for common use cases.

Scenario 1: Muting alarm actions during scheduled maintenance windows – Regular maintenance activities that occur on a predictable schedule, such as system or database updates when services are intentionally unavailable or operating in degraded mode.

- Cron expression `0 2 * * SUN` with duration `PT4H` - Mutes alarms every Sunday from 2:00 AM to 6:00 AM for weekly system maintenance.
- Cron expression `0 1 1 * *` with duration `PT6H` - Mutes alarms on the first day of each month from 1:00 AM to 7:00 AM for monthly database maintenance.

Scenario 2: Muting non-critical alarms during non-business hours – Reducing alert fatigue during weekends or holidays when immediate attention is not required.

- Cron expression `0 18 * * FRI` with duration `P2DT12H` - Mutes alarms every weekend from Friday 6:00 PM to Monday 6:00 AM.

Scenario 3: Muting performance alarms during daily backup operations – Daily automated backup processes that temporarily increase resource utilization and may trigger performance-related alarms during predictable time windows.

- Cron expression `0 23 * * *` with duration `PT2H` - Mutes alarms every day from 11:00 PM to 1:00 AM during nightly backup operations that temporarily increase disk I/O and CPU utilization.

Scenario 4: Muting duplicate alarms during active troubleshooting sessions – Temporary muting of alarm actions while teams are actively investigating and resolving issues, preventing notification noise and allowing focused problem resolution.

- At expression `at(2024-05-10T14:00)` with duration `PT4H` - Mutes alarms on May 10, 2024 from 2:00 PM to 6:00 PM during an active incident response session.

Scenario 5: Muting alarm actions during planned company shutdowns – One-time extended maintenance periods or company-wide shutdowns where all systems are intentionally offline for extended periods.

- At expression `at(2024-12-23T00:00)` with duration `P7D` - Mutes alarms for the entire week of December 23-29, 2024 during annual company shutdown.

Limits

General CloudWatch quotas

For information about general CloudWatch service quotas that apply to alarms, see CloudWatch service quotas.

Limits that apply to alarms based on metric math expressions

Alarms based on metric math expressions can reference a maximum of 10 metrics. This is a hard limit that cannot be increased. If you need to monitor more than 10 metrics in a single alarm, consider one of the following approaches:

- If the metrics are in the same namespace, use a Metrics Insights query in your alarm instead of a metric math expression. Metrics Insights can aggregate across many metrics with a single query.
- Pre-aggregate metrics into custom metrics using a Lambda function, then reference the aggregated metrics in your alarm expression.
- Split your logic across multiple alarms and combine them using a composite alarm.

Limits that apply to alarms based on Metrics Insights queries

When working with CloudWatch Metrics Insights alarms, be aware of these functional limits:

- A default of 200 alarms using the Metrics Insights query per account per Region
- Only the latest 3 hours of data can be used for evaluating the alarm's conditions. However, you can visualize up to two weeks of data on the alarm's detail page graph
- Alarms evaluating multiple time series will limit the number of contributors in ALARM to 100

- Assuming the query retrieves 150 time series:
  - If there are fewer than 100 contributors in ALARM (for example 95), the `StateReason` will be "95 out of 150 time series evaluated to ALARM"
  - If there are more than 100 contributors in ALARM (for example 105), the `StateReason` will be "100+ time series evaluated to ALARM"
- Furthermore, if the volume of attributes is too large, the number of contributors in ALARM can be limited to less than 100.
- Metrics Insights limits on the maximum number of time series analyzed or returned apply
- During alarm evaluation, the `EvaluationState` will be set to `PARTIAL_DATA` for the following limits:
  - If the Metrics Insights query returns more than 500 time series.
  - If the Metrics Insights query matches more than 10,000 metrics.

For more information on CloudWatch service quotas and limits, see CloudWatch Metrics Insights service quotas.

Limits that apply to Log Alarms

When working with CloudWatch Log Alarms, be aware of the following limits:

Resource	Limit	Adjustable
PutLogAlarm requests per second	3 (burst: 5)	No
Maximum Log Alarms per account	~1000 (limited by CloudWatch Logs AWS managed Scheduled Query quota)	No
Maximum contributors per query execution	500	No
Maximum contributors tracked in ALARM	100	No
Maximum fields in BY clause	5	No
Maximum log lines in SNS email notification	50 (also limited by SNS notification payload limit of 256 KB)	No

- Alarms evaluating multiple contributors limit the number of contributors in ALARM to 100.
  - If there are fewer than 100 contributors in ALARM (for example 95), the `StateReason` is "95 out of 100 contributors evaluated to ALARM."
  - If there are more than 100 contributors in ALARM (for example 105), the `StateReason` is "100+ contributors evaluated to ALARM."
  - If the volume of attributes is too large, the number of contributors in ALARM can be limited to fewer than 100.
- During alarm evaluation, the `EvaluationState` is set to `PARTIAL_DATA` if the query returns more than 500 contributor groups.
- The total log lines included in a notification are limited by the requested count, the total results available, and the SNS payload size limit. If the log lines exceed the payload limit, fewer lines are included.

Limits that apply to alarms based on PromQL queries

When working with CloudWatch alarms that use Prometheus Query Language (PromQL) queries, be aware of these functional limits:

- Your account can have up to 15,000 PromQL alarms in each AWS Region by default.
- Alarms evaluating multiple time series will limit the number of contributors in ALARM to 100.
  - If there are fewer than 100 contributors in ALARM (for example 95), the `StateReason` will be "95 time series evaluated to ALARM"
  - If there are more than 100 contributors in ALARM (for example 105), the `StateReason` will be "100+ time series evaluated to ALARM"
  - Furthermore, if the volume of attributes is too large, the number of contributors in ALARM can be limited to less than 100.
- PromQL query limits apply to the maximum number of time series that are analyzed or returned.
- During alarm evaluation, the `EvaluationState` will be set to `PARTIAL_DATA` if the PromQL query returns more than 500 time series.

Limits that apply to alarms based on connected data sources

- When CloudWatch evaluates an alarm, it does so every minute, even if the period for the alarm is longer than one minute. For the alarm to work, the Lambda function must be able to return

a list of timestamps starting on any minute, not only on multiples of the period length. These timestamps must be spaced one period length apart.

Therefore, if the data source queried by the Lambda can only return timestamps that are multiples of the period length, the function should "re-sample" the fetched data to match the timestamps expected by the `GetMetricData` request.

For example, an alarm with a five-minute period is evaluated every minute using five-minute windows that shift by one minute each time. In this case:

- For the alarm evaluation at 12:15:00, CloudWatch expects data points with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`.
- Then for the alarm evaluation at 12:16:00, CloudWatch expects data points with timestamps of `12:01:00`, `12:06:00`, and `12:11:00`.

- When CloudWatch evaluates an alarm, any data points returned by the Lambda function that don't align with the expected timestamps are dropped, and the alarm is evaluated using the remaining expected data points. For example, when the alarm is evaluated at `12:15:00` it expects data with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`. If it receives data with timestamps of `12:00:00`, `12:05:00`, `12:06:00`, and `12:10:00`, the data from `12:06:00` is dropped and CloudWatch evaluates the alarm using the other timestamps.

  Then for the next evaluation at `12:16:00`, it expects data with timestamps of `12:01:00`, `12:06:00`, and `12:11:00`. If it only has the data with timestamps of `12:00:00`, `12:05:00`, and `12:10:00`, all of these data points are ignored at 12:16:00 and the alarm transitions into the state according to how you specified the alarm to treat missing data. For more information, see Alarm evaluation.
- We recommend that you create these alarms to take actions when they transition to the `INSUFFICIENT_DATA` state, because several Lambda function failure use cases will transition the alarm to `INSUFFICIENT_DATA` regardless of the way that you set the alarm to treat missing data.
- If the Lambda function returns an error:
  - If there is a permission problem with calling the Lambda function, the alarm begins having missing data transitions according to how you specified the alarm to treat missing data when you created it.
  - Any other error coming from the Lambda function causes the alarm to transition to `INSUFFICIENT_DATA`.

- If the metric requested by the Lambda function has some delay so that the last data point is always missing, you should use a workaround. You can create an M out of N alarm or increase the evaluation period of the alarm. For more information about M out of N alarms, see Alarm evaluation.

Getting started

Topics

- Create alarms
- Acting on alarm changes
- Configure alarm mute rules

Create alarms

Topics

- Create an alarm using a PromQL query
- Create a CloudWatch alarm based on a static threshold
- Create a CloudWatch alarm based on a metric math expression
- Create a CloudWatch alarm based on anomaly detection
- Create an alarm based on Metrics Insights
- Create an alarm based on a connected data source
- Create a metric alarm that uses a wall clock evaluation window
- Create an alarm that uses a warm-up period
- Alarming on logs
- Create a composite alarm

Create an alarm using a PromQL query

You can create a CloudWatch alarm that uses a PromQL instant query to monitor metrics ingested through the CloudWatch OTLP endpoint. All matching time series returned by the query are considered breaching, and the alarm tracks each breaching time series as a contributor. For more information about how PromQL alarms work, see PromQL alarms.

Creating a PromQL alarm using the AWS Management Console

This example creates an alarm that monitors the HTTP 5xx server-error rate for the checkout-api service and alerts you when the error rate exceeds 5 percent.

To create a PromQL alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **PromQL** for the metric type.
5. In **Editor** mode, enter the PromQL query:

```
sum(rate({"http.server.request.count", "@resource.service.name"="checkout-api",
 "http.response.status_code"=~"5.."}[5m])) / sum(rate({"http.server.request.count",
 "@resource.service.name"="checkout-api"}[5m])) * 100 > 5
```

6. Under **Conditions**, specify the following:
   - For **Evaluation interval**, choose **`1 minute`**. This value defines how often CloudWatch evaluates the PromQL query.
   - For **Pending period**, enter **`120`**. This value is the number of seconds that a contributor must breach before it enters the ALARM state.
   - For **Recovery period**, enter **`300`**. This value is the number of seconds that a contributor must stop breaching before it enters the OK state.
7. Configure notifications and actions as needed.
8. Add a name and description for your alarm.
9. Choose **Next**.
10. Choose **Create alarm**.

Creating a PromQL alarm (AWS CLI)

Use the PutMetricAlarm API action to create a PromQL alarm.

Example: PromQL alarm that triggers when the 5xx error rate for a service exceeds 5 percent

This alarm notifies an Amazon SNS topic when the 5xx error rate exceeds 5 percent. It waits 60 seconds (the pending period) before entering the `ALARM` state, and 300 seconds (the recovery period) before returning to `OK`.

```
aws cloudwatch put-metric-alarm \
  --alarm-name MyPromQLAlarm \
  --evaluation-criteria '{"PromQLCriteria":
{"Query":"sum(rate({\"http.server.request.count\", \"@resource.service.name
\"=\"checkout-api\", \"http.response.status_code\"=~\"5..\"}[5m])) /
 sum(rate({\"http.server.request.count\", \"@resource.service.name\"=\"checkout-api\"}
[5m])) * 100 > 5","PendingPeriod":60,"RecoveryPeriod":300}}' \
  --evaluation-interval 60 \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:MyTopic
```

Example: PromQL alarm for p99 request latency with a pending period

This alarm waits 300 seconds (5 minutes) before transitioning to the `ALARM` state, and waits 600 seconds (10 minutes) before recovering.

```
aws cloudwatch put-metric-alarm \
  --alarm-name HighLatencyAlarm \
  --evaluation-criteria '{"PromQLCriteria":{"Query":"histogram_quantile(0.99,
 {\"http.server.request.duration\", \"@resource.service.name\"=\"checkout-api\"}) >
 0.5","PendingPeriod":300,"RecoveryPeriod":600}}' \
  --evaluation-interval 60
```

Example Create a PromQL alarm with an SNS notification action

```
aws cloudwatch put-metric-alarm \
  --alarm-name MyPromQLAlarmWithAction \
  --evaluation-criteria '{"PromQLCriteria":{"Query":"avg by (\"@resource.k8s.pod.name
\") (rate(container_cpu_usage_seconds_total{\"@resource.k8s.namespace.name\"=
\"production\"}[5m])) * 100 > 80","PendingPeriod":0,"RecoveryPeriod":0}}' \
  --evaluation-interval 60 \
  --alarm-actions arn:aws:sns:us-east-1:123456789012:MyTopic
```

Example: PromQL anomaly-detection alarm using a statistical band

A PromQL alarm can detect anomalies without a separate anomaly detector. Instead of the `ANOMALY_DETECTION_BAND` metric-math function that standard metric alarms use, you build

the expected range directly in the query with functions such as `quantile_over_time` and `stddev_over_time`. The following query alerts when the request count for the checkout-api service moves more than three standard deviations from its one-hour median in either direction. The `clamp_min` function keeps the lower bound from going negative.

```
aws cloudwatch put-metric-alarm \
  --alarm-name RequestCountAnomaly \
  --evaluation-criteria '{"PromQLCriteria":{"Query":"1 * {\"http.server.request.count
\", \"@resource.service.name\"=\"checkout-api\"} > quantile_over_time(0.5,
 {\"http.server.request.count\", \"@resource.service.name\"=\"checkout-api
\"}[60m] offset 1m) + 3 * stddev_over_time({\"http.server.request.count
\", \"@resource.service.name\"=\"checkout-api\"}[60m] offset 1m) or 1 *
 {\"http.server.request.count\", \"@resource.service.name\"=\"checkout-
api\"} < clamp_min(quantile_over_time(0.5, {\"http.server.request.count
\", \"@resource.service.name\"=\"checkout-api\"}[60m] offset 1m) - 3 *
 stddev_over_time({\"http.server.request.count\", \"@resource.service.name\"=
\"checkout-api\"}[60m] offset 1m), 0)"}}' \
  --evaluation-interval 60
```

Example: PromQL alarm that triggers an Application Auto Scaling step scaling policy

You can attach a PromQL alarm to an Application Auto Scaling step scaling policy. First, create the step scaling policy with the `put-scaling-policy` command. The command returns a policy ARN.

```
aws application-autoscaling put-scaling-policy \
  --service-namespace ecs \
  --scalable-dimension ecs:service:DesiredCount \
  --resource-id service/my-cluster/my-service \
  --policy-name promql-5xx-scale-out \
  --policy-type StepScaling \
  --step-scaling-policy-configuration
 '{"AdjustmentType":"ChangeInCapacity","StepAdjustments":
[{"MetricIntervalLowerBound":0,"ScalingAdjustment":1}],"Cooldown":60}'
```

Then create the PromQL alarm, setting `--alarm-actions` to the policy ARN from the previous output. This alarm scales out the service when the 5xx error rate for the checkout-api service exceeds 5 percent. When a PromQL alarm triggers an Auto Scaling action, the query must return a single time series and include a comparison. Wrap the expression in an aggregation function such as `avg`, `sum`, `max`, or `min`, followed by the comparison, as shown in the following command.

```
aws cloudwatch put-metric-alarm \
```

```
  --alarm-name checkout-5xx-scale-out \
  --evaluation-criteria '{"PromQLCriteria":
{"Query":"avg(sum(rate({\"http.server.request.count\", \"@resource.service.name
\"=\"checkout-api\", \"http.response.status_code\"=~\"5..\"}[5m])) /
 sum(rate({\"http.server.request.count\", \"@resource.service.name\"=\"checkout-api\"}
[5m])) * 100) > 5","PendingPeriod":60,"RecoveryPeriod":300}}' \
  --evaluation-interval 60 \
  --alarm-actions arn:aws:autoscaling:us-
east-1:123456789012:scalingPolicy:abcd1234-5678-90ab-cdef-example11111:resource/ecs/
service/my-cluster/my-service:policyName/promql-5xx-scale-out
```

Creating a PromQL alarm from Query Studio

This example shows how to create a PromQL alarm from Query Studio that alerts you when the average HTTP request duration for a service exceeds 500 milliseconds.

Unlike standard CloudWatch alarms where the threshold is configured as a separate step, PromQL alarms define the alarm condition (threshold) as part of the query itself. For example, the comparison operator (>) and threshold value (`0.5`) are embedded directly in the PromQL expression.

To create a PromQL alarm from Query Studio

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane below **Metrics**, choose **Query Studio**.
3. Select **PromQL** from the query language drop-down menu.
4. Build your query using one of the following modes:
   - In **Builder** mode, select a metric name from the **Metric** field (for example, `http.server.request.duration`). Add label filters as needed (for example, `@resource.service.name` = `my-api`). To define the alarm threshold, select a **Basic Operation** (for example, >) and enter a **Number** (for example, `0.5`).
   - In **Code** mode, enter the PromQL expression directly, for example:

     ```
     histogram_avg({"http.server.request.duration", "@resource.service.name"="my-
     api"}) > 0.5
     ```
5. Choose **Run** to execute the query and verify it returns the expected results.
6. Choose **Create alarm** from the actions menu.

7. You are redirected to the CloudWatch alarm creation page with your PromQL query pre-populated.
8. Under **Conditions**, specify the following:
   - For **Evaluation interval**, choose **`1 minute`**, to define how often the PromQL query is evaluated.
   - For **Pending period**, enter **`60`**, duration in seconds the query must be breaching before entering ALARM state. This means the latency must exceed the threshold for at least 60 seconds before the alarm fires.
   - For **Recovery period**, enter **`120`**, duration in seconds the query must not be breaching before entering OK state. This means the latency must stay below the threshold for at least 120 seconds before the alarm recovers.
9. Configure notifications and actions as needed.
10. Add a name and description for your alarm.
11. Choose **Next**.
12. Choose **Create alarm**.

> **Note**
>
> The PromQL query must return a single time series to create an alarm. If your query returns multiple time series, use aggregation functions such as `sum`, `avg`, or `topk` to reduce the result to a single series before creating the alarm.

Create a CloudWatch alarm based on a static threshold

You choose a CloudWatch metric for the alarm to watch, and the threshold for that metric. The alarm goes to `ALARM` state when the metric breaches the threshold for a specified number of evaluation periods.

If you are creating an alarm in an account set up as a monitoring account in CloudWatch cross-account observability, you can set up the alarm to watch a metric in a source account linked to this monitoring account. For more information, see CloudWatch cross-account observability.

To create an alarm based on a single metric

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
   - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
   - In the search box, enter the name of a metric, account ID, account label, dimension, or resource ID. Then, choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.
6. Choose the **Graphed metrics** tab.
   a. Under **Statistic** , choose one of the statistics or predefined percentiles, or specify a custom percentile (for example, **p95.45**).
   b. Under **Period**, choose the evaluation period for the alarm. When evaluating the alarm, each period is aggregated into one data point.

      You can also choose whether the y-axis legend appears on the left or right while you're creating the alarm. This preference is used only while you're creating the alarm.
   c. Choose **Select metric**.

      The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.
7. Under **Conditions**, specify the following:
   a. For **Whenever *`metric`* is**, specify whether the metric must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
   b. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

      To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.

c. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

d. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see Percentile-based alarms and low data samples.

8. Choose **Next**.

9. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

   To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

   In CloudWatch cross-account observability, you can choose to have notifications sent to multiple AWS accounts. For example, to both the monitoring account and the source account.

   To have the alarm not send notifications, choose **Remove**.

10. To have the alarm perform Auto Scaling, EC2, Lambda, investigation, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. Alarms can perform Systems Manager actions and investigation actions only when they go into ALARM state. For more information about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    To have the alarm start an investigation, choose **Add investigation action** and then select your investigation group. For more information about , see CloudWatch investigations.

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

11. When finished, choose **Next**.

12. Enter a name and description for the alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting,

which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources. Then choose **Next**.

13. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see Adding an alarm to a CloudWatch dashboard.

Create a CloudWatch alarm based on a metric math expression

Metric alarms are designed to evaluate time series that you define from either a single metric, or a metric math expression that combines or transforms one or more metrics into a time series that deliver insights more closely aligned to your unique needs. To create an alarm based on a metric math expression, choose one or more CloudWatch metrics to use in the expression. Then, specify the expression, threshold, and evaluation periods.

You can't create an alarm based on the **SEARCH** expression. Only alarms based on Metrics Insights SQL queries can operate on multiple time series.

To create an alarm that's based on a metric math expression

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**, and then perform one of the following actions:
   - Select a namespace from the **AWS namespaces** dropdown or **Custom namespaces** dropdown. After you select a namespace, you continue choosing options until a list of metrics appears, where you select the checkbox next to the correct metric.
   - Use the search box to find a metric, account ID, dimension, or resource ID. After you enter the metric, dimension, or resource ID, you continue choosing options until a list of metrics appears, where you select the check box next to the correct metric.
5. (Optional) If you want to add another metric to a metric math expression, you can use the search box to find a specific metric. You can add as many as 10 metrics to a metric math expression.
6. Select the **Graphed metrics** tab. For each of the metrics that you previously added, perform the following actions:

a. Under the **Statistic** column, select the dropdown menu. In the dropdown menu, choose one of the predefined statistics or percentiles. Use the search box in the dropdown menu to specify a custom percentile.

b. Under the **Period** column, select the dropdown menu. In the dropdown menu, choose one of the predefined evaluation periods.

   While you're creating your alarm, you can specify whether the Y-axis legend appears on the left or right side of your graph.

> **Note**
>
> When CloudWatch evaluates alarms, periods are aggregated into single data points.

7. Choose the **Add math** dropdown, and then select **Start with an empty expression** from the list of predefined metric math expressions.

   After you choose **Start with an empty expression**, a math expression box appears where you apply or edit math expressions.

8. In the math expression box, enter your math expression, and then choose **Apply**.

   After you choose **Apply**, an **ID** column appears next to the **Label** column.

   To use a metric or the result of another metric math expression as part of your current math expression's formula, you use the value that's shown under the **ID** column. To change the value of **ID**, you select the pen-and-paper icon next to the current value. The new value must begin with a lowercase letter and can include numbers, letters, and the underscore symbol. Changing the value of **ID** to a more significant name can make your alarm graph easier to understand.

   For information about the functions that are available for metric math, see Metric math syntax and functions.

9. (Optional) Add more math expressions, using both metrics and the results of other math expressions in the formulas of the new math expressions.

10. When you have the expression to use for the alarm, clear the check boxes to the left of every other expression and every metric on the page. Only the check box next to the expression to use for the alarm should be selected. The expression that you choose for the alarm must produce a single time series and show only one line on the graph. Then choose **Select metric**.

The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

11. For **Whenever *`expression`* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.
12. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

    To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.
13. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
14. Choose **Next**.
15. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.
16. To have the alarm perform Auto Scaling, Amazon EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

> **Note**
>
> To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

17. When finished, choose **Next**.
18. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
19. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

You can also add alarms to a dashboard. For more information, see Adding an alarm to a CloudWatch dashboard.

Create a CloudWatch alarm based on anomaly detection

You can create an alarm based on CloudWatch anomaly detection, which analyzes past metric data and creates a model of expected values. The expected values take into account the typical hourly, daily, and weekly patterns in the metric.

You set a value for the anomaly detection threshold, and CloudWatch uses this threshold with the model to determine the "normal" range of values for the metric. A higher value for the threshold produces a thicker band of "normal" values.

You can choose whether the alarm is triggered when the metric value is above the band of expected values, below the band, or either above or below the band.

You also can create anomaly detection alarms on single metrics and the outputs of metric math expressions. You can use these expressions to create graphs that visualize anomaly detection bands.

In an account set up as a monitoring account for CloudWatch cross-account observability, you can create anomaly detectors on metrics in source accounts in addition to metrics in the monitoring account.

For more information, see Using CloudWatch anomaly detection.

> **Note**
>
> If you're already using anomaly detection for visualization purposes on a metric in the Metrics console and you create an anomaly detection alarm on that same metric, then the

threshold that you set for the alarm doesn't change the threshold that you already set for visualization. For more information, see Creating a graph.

To create an alarm that's based on anomaly detection

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Do one of the following:
   - Choose the service namespace that contains your metric, and then continue choosing options as they appear to narrow down your options. When a list of metrics appears, select the check box that's next to your metric.
   - In the search box, enter the name of a metric, dimension, or resource ID. Select one of the results, and then continue choosing options as they appear until a list of metrics appears. Select the check box that's next to your metric.
6. Choose **Graphed metric**.
   a. (Optional) For *Statistic*, choose the dropdown, and then select one of the predefined statistics or percentiles. You can use the search box in the dropdown to specify a custom percentile, such as **p95.45**.
   b. (Optional) For *Period*, choose the dropdown, and then select one of the predefined evaluation periods.

   > **Note**
   >
   > When CloudWatch evaluates your alarm, it aggragates the period into a single datapoint. For an anomaly detection alarm, the evaluation period must be one minute or longer.

7. Choose **Next**.
8. Under ***Conditions***, specify the following:
   a. Choose **Anomaly detection**.

If the model for this metric and statistic already exists, CloudWatch displays a preview of the anomaly detection band in the graph at the top of the screen. After you create your alarm, it can take up to 15 minutes for the actual anomaly detection band to appear in the graph. Before that, the band that you see is an approximation of the anomaly detection band.

> **Tip**
>
> To see the graph at the top of the screen in a longer time frame, choose **Edit** at the top-right of the screen.

If the model for this metric and statistic doesn't already exist, CloudWatch generates the anomaly detection band after you finish creating your alarm. For new models, it can take up to 3 hours for the actual anomaly detection band to appear in your graph. It can take up to two weeks for the new model to train, so the anomaly detection band shows more accurate expected values.

b. For **Whenever *`metric`* is**, specify when to trigger the alarm. For example, when the metric is greater than, lower than, or outside the band (in either direction).

c. For **Anomaly detection threshold**, choose the number to use for the anomaly detection threshold. A higher number creates a thicker band of "normal" values that is more tolerant of metric changes. A lower number creates a thinner band that will go to `ALARM` state with smaller metric deviations. The number does not have to be a whole number.

d. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

   To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see Alarm evaluation.

e. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

f. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates.

If you choose **Ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see Percentile-based alarms and low data samples.

9. Choose **Next**.
10. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.

    To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    Choose **Remove** if you don't want the alarm to send notifications.
11. You can set up the alarm to perform EC2 actions or invoke a Lambda function when it changes state, or to create a Systems Manager OpsItem or incident when it goes into ALARM state. To do this, choose the appropriate button and then choose the alarm state and action to perform.

    If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    For more information about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    > **Note**
    >
    > To create an alarm that performs an AWS Systems Manager Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.
12. Choose **Next**.
13. Under ***Name and description***, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

> **Tip**
>
> The alarm name must contain only UTF-8 characters, and can't contain ASCII control characters

14. Under ***Preview and create***, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Creating an anomaly detection alarm using the AWS CLI

You can also create an anomaly detection alarm with the AWS CLI. First create the anomaly detector, and then create an alarm that uses an `ANOMALY_DETECTION_BAND` metric math expression to define the expected range.

To create an anomaly detection alarm

1. Create the anomaly detector using the put-anomaly-detector command.

```
aws cloudwatch put-anomaly-detector --single-metric-anomaly-detector
 '{"Namespace":"AWS/EC2","MetricName":"CPUUtilization","Dimensions":
[{"Name":"InstanceId","Value":"instance-id"}],"Stat":"Average"}'
```

   If the command succeeds, it returns no output.

2. Create the alarm with a metric-math band using the put-metric-alarm command. The `ANOMALY_DETECTION_BAND` expression defines the expected range.

```
aws cloudwatch put-metric-alarm --alarm-name cpu-anomaly --comparison-
operator LessThanLowerOrGreaterThanUpperThreshold --evaluation-periods 1
 --threshold-metric-id ad1 --metrics '[{"Id":"m1","MetricStat":{"Metric":
{"Namespace":"AWS/EC2","MetricName":"CPUUtilization","Dimensions":
[{"Name":"InstanceId","Value":"instance-
id"}]},"Period":300,"Stat":"Average"},"ReturnData":true},
{"Id":"ad1","Expression":"ANOMALY_DETECTION_BAND(m1, 2)","ReturnData":true}]'
```

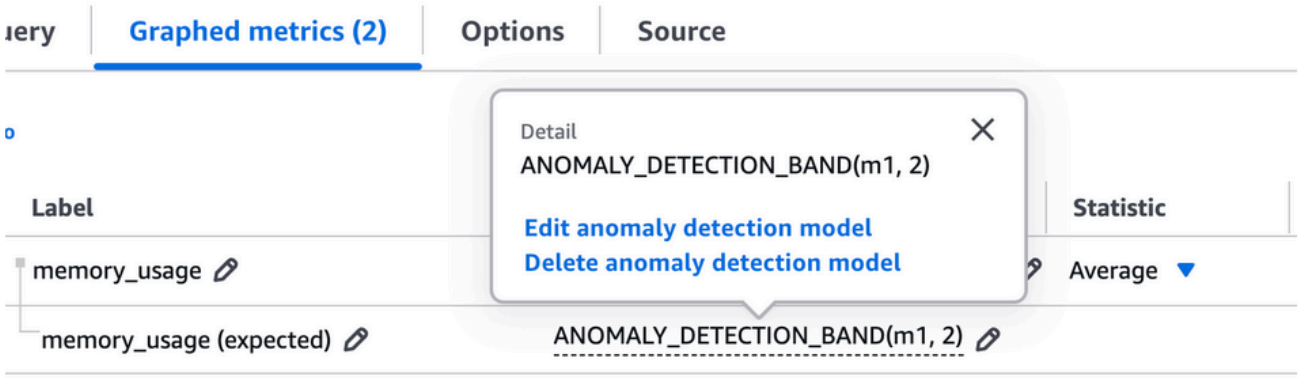
Editing an anomaly detection model

After you create an alarm, you can adjust the anomaly detection model. You can exclude certain time periods from being used in the model creation. It is critical that you exclude unusual events

such as system outages, deployments, and holidays from the training data. You can also specify whether to adjust the model for Daylight Savings Time changes.

To edit the anomaly detection model for an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose the name of the alarm. If necessary, use the search box to find the alarm.
4. Choose **View**, **In metrics**.
5. In the **Details** column, choose the **ANOMALY_DETECTION_BAND** keyword, and then choose **Edit anomaly detection model** in the popup.



6. To exclude a time period from being used to produce the model, choose the calendar icon by **End date**. Then, select or enter the days and times to exclude from training and choose **Apply**.
7. If the metric is sensitive to Daylight Savings Time changes, select the appropriate time zone in the **Metric timezone** box.
8. Choose **Update**.

Deleting an anomaly detection model

Using anomaly detection for an alarm accrues charges. As a best practice, if your alarm no longer needs an anomaly detection model, delete the alarm first and the model second. When anomaly detection alarms are evaluated, any missing anomaly detectors are created on your behalf. If you delete the model without deleting the alarm, the alarm automatically recreates the model.

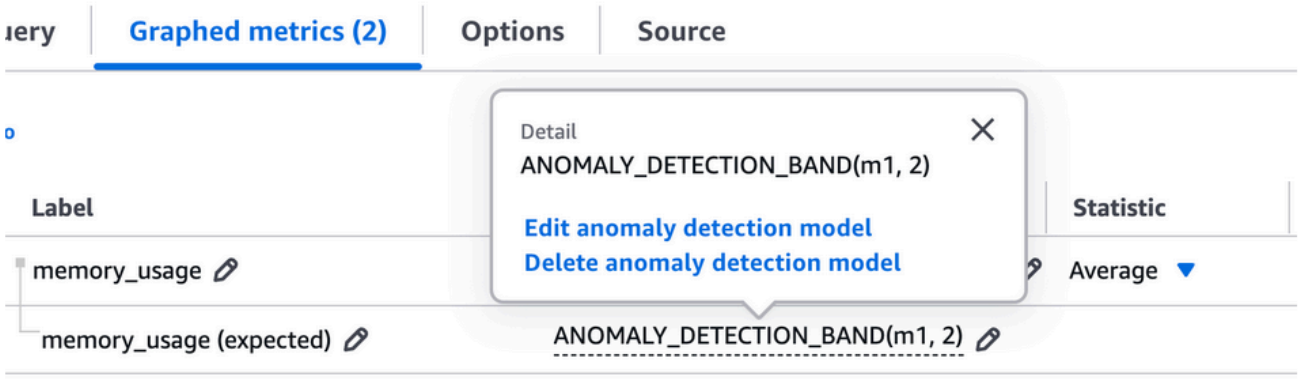
To delete an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose the name of the alarm.
4. Choose **Actions**, **Delete**.
5. In the confirmation box, choose **Delete**.

To delete an anomaly detection model that was used for an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**.
3. Choose **Browse**, and then select the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.
   - (Optional) If you're using the original interface, choose **All metrics**, and then choose the metric that includes the anomaly detection model. You can search for your metric in the search box or select your metric by choosing through the options.
4. Choose **Graphed metrics**.
5. In the **Graphed metrics** tab, in the **Details** column, choose the **ANOMALY_DETECTION_BAND** keyword, and then choose **Delete anomaly detection model** in the popup.



   - (Optional) If you're using the original interface, choose **Edit model**. You're directed to a new screen. On the new screen, choose **Delete model**, and then choose **Delete**.

Improving anomaly detection alarm accuracy with feedback

After you create an anomaly detection alarm, you can provide feedback to help CloudWatch improve the accuracy of the anomaly detection model. CloudWatch uses your feedback to adjust

how the model learns from your metric's behavior, which can result in more accurate anomaly detection bands over time.

You can provide feedback when you notice that:

- The alarm triggered correctly, and you want to confirm that the detection was accurate.
- The alarm triggered, but the metric behavior was actually normal (a false alarm).
- The alarm did not trigger, but the metric behavior was anomalous (a missed detection).

To submit feedback on an anomaly detection alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**.
3. Choose the name of the anomaly detection alarm that you want to provide feedback for.
4. Choose the **Improve this alarm** button.
5. In the feedback dialog box, specify a **Start time** and **End time** for the time range that your feedback applies to. This should cover the period where you observed the alarm behavior you want to report.
6. For **Feedback type**, choose one of the following:

   - **Correct** – The alarm correctly detected an anomaly (or correctly did not trigger). This confirms that the model is performing well for this time range. No model adjustments are made.
   - **False alarm** – The alarm triggered, but the metric behavior was actually normal. CloudWatch adjusts the anomaly detection model to reduce false positives.
   - **Missed** – The alarm did not trigger, but the metric behavior was anomalous and should have been detected. CloudWatch adjusts the anomaly detection model to improve sensitivity.
7. For **Reason**, choose the reason that best describes why the alarm behavior was incorrect:

   - **Bands learning too fast** – The anomaly detection bands are adapting too quickly to recent changes in metric behavior, treating new patterns as normal too soon.
   - **Bands learning too slow** – The anomaly detection bands are not adapting quickly enough to legitimate changes in metric behavior.
   - **Bands too wide** – The expected range of normal values is too broad, causing the alarm to miss anomalies.

- **Bands too narrow** – The expected range of normal values is too tight, causing the alarm to trigger on normal metric variations.
- **Band level off** – The anomaly detection bands are not aligned with the expected baseline level of the metric.
- **Planned deployment** – The alarm fired due to a planned deployment or change.
- **Scaling event** – The alarm fired due to an expected scaling event (such as auto-scaling).
- **Other** – The alarm behavior was incorrect for a reason not listed above.

8. Choose **Submit**.

CloudWatch processes your feedback and adjusts the anomaly detection model accordingly. For false alarm and missed feedback, CloudWatch automatically tunes the model's learning rate or band width based on the reason that you provide.

> **Note**
>
> You can submit feedback multiple times for different time ranges. Each piece of feedback is processed independently and contributes to improving the model's accuracy over time.

> **Note**
>
> When you submit feedback, the CloudWatch console calls the `SubmitAlarmFeedback` API on your behalf. This API is used by the CloudWatch console only and is not intended to be called directly.

Create an alarm based on Metrics Insights

You can create an alarm that monitors multiple time series across a fleet of resources. Unlike single-instance alarms that trigger actions on individual instances, fleet monitoring alarms let you aggregate metrics across multiple resources and trigger based on fleet-wide conditions.

Setting up a Multi Time Series alarm using the AWS Management Console

This example shows how to create an alarm that monitors memory utilization across a fleet of instances and alerts you when more than two instances exceed a threshold.

To create a multi time series alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. Under **Metrics**, enter a Metrics Insights query:

```
SELECT MAX(mem_used_percent)
FROM "CWAgent"
GROUP BY InstanceId
ORDER BY MAX() DESC
```

6. Choose **Next**.
7. Under **Conditions**, specify the following:
   - For **Threshold type**, choose **Static**.
   - For **When metric is**, choose **Greater than** and enter **80**.
   - For **Datapoints to alarm**, enter **2**.
8. Configure notifications and actions as needed.
9. Add a name and description for your alarm.
10. Choose **Create alarm**.

This alarm differs from single-instance alarms in several ways:

- It monitors multiple time series simultaneously through the use of a metrics query. The metrics query is refreshed every time the alarm evaluates, thus your alarm automatically adapts as resources are created, paused, or deleted.
- For each contributor that breaches the threshold, the alarm sends a contributor state change event, which has a different event type in EventBridge than an alarm state change event. The alarm itself also changes state: as soon as at least one contributor is in alarm, the alarm also enters the alarm state.
- Some actions however, such as SSM Incident, are triggered at the alarm level. Such actions are not repeated when the list of contributors in alarm changes.

This alarm differs from aggregated metric-query alarms in several ways:

- It monitors time series individually instead of an aggregate, using the `GROUP BY` clause.
- It follows the level of granularity that you set according to your needs: for example, it can alarm on every Amazon EC2 instance (most granular level of Amazon EC2 metrics) or per Amazon RDS table (aggregated across various operations on a table), depending on which fields you set in the `GROUP BY` clause
- It prioritizes evaluation using the `ORDER BY` clause.
- For each contributor that breaches the threshold, the alarm sends a contributor state change event, which has a different event type in EventBridge than an alarm state change event. The alarm itself also changes state: as soon as at least one contributor is in alarm, the alarm also enters the alarm state.
- Some actions however, such as SSM Incident, are triggered at the alarm level. Such actions are not repeated when the list of contributors in alarm changes.

Create an alarm based on a connected data source

You can create alarms that watch metrics from data sources that aren't in CloudWatch. For more information about creating connections to these other data sources, see Query metrics from other data sources.

To create an alarm on metrics from a data source that you have connected to

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Choose the **Multi source query** tab.
4. For **Data source**, select the data source that you want to use.
5. The query builder prompts you for the information necessary for the query to retrieve the metrics to use for the alarm. The workflow is different for each data source, and is tailored to the data source. For example, for Amazon Managed Service for Prometheus and Prometheus data sources, a PromQL query editor box with a query helper appears.
6. When you have finished constructing the query, choose **Graph query**.
7. If the sample graph looks the way that you expect, choose **Create alarm**.

8. The **Specify metric and conditions** page appears. If the query you are using produces more than one time series, you see a warning banner at the top of the page. If you do, select a function to use to aggregate the time series in **Aggregation function**.
9. (Optional) Add a **Label** for the alarm.
10. For **Whenever *`your-metric-name`* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal**, or **Lower**. Then for **than . . .**, specify a number for your threshold value.
11. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the `ALARM` state to trigger the alarm. If the two values here match, you create an alarm that goes to `ALARM` state if that many consecutive periods are breaching.

    To create an M out of N alarm, specify a number for the first value that is lower than the number for the second value. For more information, see Alarm evaluation.
12. For **Missing data treatment**, choose how the alarm behaves when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.
13. Choose **Next**.
14. For **Notification**, specify an Amazon SNS topic to notify when your alarm transitions to the `ALARM`, `OK`, or `INSUFFICIENT_DATA` state.

    a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    > **Note**
    >
    > We recommend that you set the alarm to take actions when it goes into **Insufficient data** state in addition to when it goes into **Alarm** state. This is because many issues with the Lambda function that connects to the data source can cause the alarm to transition to **Insufficient data**.

    b. (Optional) To not send Amazon SNS notifications, choose **Remove**.
15. To have the alarm perform Auto Scaling, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

> **Note**
>
> To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

16. Choose **Next**.
17. Under **Name and description**, enter a name and description for your alarm, and choose **Next**. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

> **Tip**
>
> The alarm name must contain only UTF-8 characters. It can't contain ASCII control characters.

18. Under **Preview and create**, confirm that your alarm's information and conditions are correct, and choose **Create alarm**.

Create a metric alarm that uses a wall clock evaluation window

You can create a metric alarm that aligns its evaluation window to wall clock boundaries instead of a sliding window. For information about how wall clock windows work and when to use them, see Wall clock window.

Creating a wall clock alarm using the AWS Management Console

This example shows how to create an alarm that uses a 1-hour wall clock window.

To create a wall clock alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**, choose the metric you want to monitor, and then choose **Select metric**.
5. Under **Metric**, for **Period**, choose a period that supports wall clock windows: **`1 minute`**, **`5 minutes`**, **`1 hour`**, **`1 day`**, or **`1 week`**.
6. Under **Conditions**, configure the threshold and comparison operator for your alarm.
7. For **Evaluation window**, choose **Wall clock window**.

> **Note**
>
> If you don't see the **Evaluation window** option, the metric type or period you selected does not support wall clock windows. Wall clock windows are not available for high-resolution alarms, composite alarms, or PromQL alarms, and are only available for periods of 1 minute, 5 minutes, 1 hour, 1 day, or 1 week.

8. Optionally, choose a time zone for **Time zone**. The alarm aligns its evaluation window to clock boundaries in this time zone. If you don't choose a time zone, CloudWatch uses UTC. For details about how time zones affect wall clock boundaries, see Time zones and daylight saving time.
9. Configure notifications and actions as needed, then choose **Next**.
10. Add a name and description for your alarm, then choose **Next**.
11. Review the alarm configuration and choose **Create alarm**.

Creating a wall clock alarm (AWS CLI)

Use the PutMetricAlarm API action to create or update a metric alarm with an `EvaluationWindow`. Set `WallClockWindow` to align the alarm to wall clock boundaries, or set `SlidingWindow` to use the default sliding behavior. If you omit the `EvaluationWindow` parameter, the alarm uses a sliding window.

Example Create an alarm that uses a 1-hour wall clock window

This alarm evaluates the most recently completed clock hour in UTC. To align to a different time zone, set `Timezone` on the `WallClockWindow`.

```
aws cloudwatch put-metric-alarm \
```

```
  --alarm-name HourlyCpuAlarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 3600 \
  --evaluation-periods 1 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --evaluation-window '{"WallClockWindow":{}}'
```

Example Create a daily alarm aligned to a specific time zone

This alarm evaluates each calendar day in the `America/New_York` time zone. Daylight saving time transitions are handled automatically.

```
aws cloudwatch put-metric-alarm \
  --alarm-name DailyBackupCheck \
  --metric-name SuccessfulBackups \
  --namespace MyApplication \
  --statistic Sum \
  --period 86400 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator LessThanThreshold \
  --evaluation-window '{"WallClockWindow":{"Timezone":"America/New_York"}}'
```

Example Create a weekly alarm in UTC

This alarm evaluates each calendar week (Monday 00:00 UTC through the following Monday 00:00 UTC).

```
aws cloudwatch put-metric-alarm \
  --alarm-name WeeklyComplianceCheck \
  --metric-name ComplianceFailures \
  --namespace MyApplication \
  --statistic Sum \
  --period 604800 \
  --evaluation-periods 1 \
  --threshold 0 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-window '{"WallClockWindow":{}}'
```

Example Switch an existing alarm back to a sliding window

Set `EvaluationWindow` to `SlidingWindow` to make the change explicit. You can also omit the parameter, which has the same effect.

```
aws cloudwatch put-metric-alarm \
  --alarm-name HourlyCpuAlarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 3600 \
  --evaluation-periods 1 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --evaluation-window '{"SlidingWindow":{}}'
```

View the evaluation window for an existing alarm

Use the DescribeAlarms API action to retrieve the evaluation window configuration of an alarm. The `EvaluationWindow` field is included in the response when the alarm uses a wall clock window. If the field is absent, the alarm uses a sliding window.

```
aws cloudwatch describe-alarms \
  --alarm-names HourlyCpuAlarm
```

You can also view the evaluation window in the AWS Management Console on the alarm details page.

Create an alarm that uses a warm-up period

Create a metric alarm or log alarm with a warm-up period to delay evaluation after creation. For more information about warm-up periods, see Alarm warm-up periods.

Set the warm-up period with the `WarmUpConfiguration` parameter on the PutMetricAlarm API. Log alarms use the PutLogAlarm API.

Creating a warm-up alarm with the AWS CLI

The following examples create and manage alarms with warm-up periods.

Example Create a metric alarm with a 30-minute warm-up period

This alarm uses default early-termination behavior: it ends warm-up and begins to evaluate as soon as enough data fills the evaluation window.

```
aws cloudwatch put-metric-alarm \
  --alarm-name MyServiceStartupAlarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --warm-up-configuration '{"WarmUpPeriodDurationInMinutes":30}'
```

Example Wait the entire warm-up period before evaluating

Set `OnlyStartEvaluatingAfterWarmUpPeriodEnds` to `true` to prevent evaluation until the full warm-up duration elapses. The alarm waits the full 60 minutes before it evaluates, even if data arrives sooner.

```
aws cloudwatch put-metric-alarm \
  --alarm-name MyServiceStartupAlarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --warm-up-configuration
 '{"WarmUpPeriodDurationInMinutes":60,"OnlyStartEvaluatingAfterWarmUpPeriodEnds":true}'
```

Example Change the warm-up period while the alarm is warming up

You can update the warm-up configuration while the alarm is still in its warm-up period. This example sets the warm-up duration to 45 minutes.

```
aws cloudwatch put-metric-alarm \
```

```
  --alarm-name MyServiceStartupAlarm \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold \
  --dimensions Name=InstanceId,Value=i-1234567890abcdef0 \
  --warm-up-configuration '{"WarmUpPeriodDurationInMinutes":45}'
```

> **Note**
>
> After the warm-up period ends, you can update the warm-up configuration. However, the update does not start a new warm-up period. The alarm continues to evaluate normally.

View the warm-up configuration

Use the DescribeAlarms API to retrieve an alarm's warm-up configuration.

```
aws cloudwatch describe-alarms \
  --alarm-names MyServiceStartupAlarm
```

During warm-up, the alarm state remains `INSUFFICIENT_DATA` and the evaluation state is `IN_WARM_UP`.

Alarming on logs

You can create CloudWatch alarms that monitor your log data in two ways:

- **Log Alarm approach** — Create a Log Alarm that runs a CloudWatch Logs Insights query on a schedule and evaluates the aggregated results directly against a threshold.
- **Metric filter approach** — Create a metric filter on a log group, then create a standard metric alarm on the resulting metric.

Create a Log Alarm

Create a Log Alarm

You can create a CloudWatch alarm that uses a CloudWatch Logs Insights query to monitor log data directly. The query runs on a schedule using a Scheduled Query, and the alarm evaluates the aggregated results against a threshold. For more information about how Log Alarms work, see Log alarms.

Prerequisites

Before you create a Log Alarm, you must create an IAM role that grants CloudWatch Logs permission to execute the scheduled query. If you also want log lines included in Amazon SNS notifications, you must create a second role.

> **Note**
>
> If you create a Log Alarm from the CloudWatch console, the console helps you create these roles.

Scheduled query execution role

The scheduled query execution role allows CloudWatch Logs to run the query on your behalf. This role is required for all Log Alarms to execute the scheduled query. The role must trust the `logs.amazonaws.com` service principal.

The following example shows the trust policy for the scheduled query execution role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "logs.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

The following example shows the permissions policy. Scope the `Resource` to the log group ARNs that the query targets.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetQueryResults",
                "logs:DescribeLogGroups"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-
group:*"
        }
    ]
}
```

Log lines role (optional)

The log lines role allows CloudWatch to fetch log lines for Amazon SNS email notifications. This role is required only if you set `ActionLogLineCount` to a value greater than 0. The role must trust the `cloudwatch.amazonaws.com` service principal.

The following example shows the trust policy for the log lines role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudwatch.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

The following example shows the permissions policy for the log lines role. The `logs:GetQueryResults` permission is required to fetch log lines. Scope the `Resource` to the log group ARNs that the query targets.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetQueryResults"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:your-log-
group:*"
        }
    ]
}
```

> **Note**
>
> To view Log Alarm query results in the CloudWatch console, the IAM user or role accessing the console must have the `logs:GetScheduledQueryData` permission. This is a console-only API and is not available through the AWS CLI or SDK. CloudTrail logs this API as a data event, so you must configure a trail with data event logging to capture it.

Creating a Log Alarm using the console

The steps in this section explain how to use the CloudWatch console to create a Log Alarm. You can also use the AWS CLI to create a Log Alarm. For more information, see Creating a Log Alarm using the AWS CLI.

To create a Log Alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**.
3. Choose **Create alarm**.
4. For **Data source**, choose **Logs**.
5. Under **Logs Query**:

a. Choose **Create query in Log Analytics**. This opens the Log Analytics page where you build your query.

b. On the Log Analytics page, write your query. When you finish, choose **Continue to Alarms** in the top right to return to the alarm console.

c. For **Aggregation expression**, specify how to aggregate the query results to return a numerical value for alarm evaluation (for example, `count(*)` or `avg(latency) by endpoint`).

6. Under **Alarm Conditions**:

a. For **Whenever the aggregated result is...**, choose a comparison operator.

b. For **than...**, enter the threshold value.

c. (Optional) Under **Additional Configuration**:

i. Modify **Datapoints to alarm** to specify the number of breaching results required to trigger `ALARM` out of last N queries.

ii. Modify **Missing data treatment** to specify how to treat missing data.

7. Under **Schedule**:

a. For **Evaluation Frequency**, choose how often the query runs (for example, every 5 minutes).

b. For **Start time offset**, the lookback window for each query execution.

c. For **End time offset** (optional), the end of the query time range as an offset in seconds from the current time.

8. Under **IAM permission**:

- Specify the role that is required to execute the scheduled query. You can either choose to create a new role or use an existing role.

9. Choose **Next**.

10. Under **Configure actions**, configure actions and notifications as needed.

- To include log lines triggering the alarm transition in Amazon SNS email notifications, under **Include query results in the actions**:

i. Add **Number of query results** to specify the number of log lines to include.

ii. Create or choose an existing role that is used to get the required query results.

11. Choose **Next**.
12. Enter a **Name** and optional **Description** (supports Markdown).
13. (Optional) Add tags.
14. Choose **Next**.
15. Under **Preview and create**, review your configuration, and then choose **Create alarm**.

Creating a Log Alarm using the AWS CLI

You can use the AWS CLI `put-log-alarm` command to create a Log Alarm.

The following example creates a Log Alarm that monitors error counts in a log group and transitions to `ALARM` state when the count exceeds 100 in 3 out of 5 query executions.

```
aws cloudwatch put-log-alarm \
    --alarm-name "HighErrorCount" \
    --alarm-description "Alarm when error count exceeds 100" \
    --comparison-operator GreaterThanThreshold \
    --threshold 100 \
    --query-results-to-evaluate 5 \
    --query-results-to-alarm 3 \
    --treat-missing-data missing \
    --alarm-actions "arn:aws:sns:us-east-1:111122223333:topic-name" \
    --scheduled-query-configuration '{
        "QueryString": "fields @timestamp, @message | filter @message like /ERROR/",
        "LogGroupIdentifiers": ["/aws/lambda/my-function"],
        "ScheduledQueryRoleARN": "arn:aws:iam::111122223333:role/ScheduledQueryRole",
        "AggregationExpression": "count(*)",
        "ScheduleConfiguration": {
            "ScheduleExpression": "rate(10 minutes)",
            "StartTimeOffset": 600
        }
    }' \
    --action-log-line-count 5 \
    --action-log-line-role-arn "arn:aws:iam::111122223333:role/LogLineRole"
```

The following table describes the key parameters for the `put-log-alarm` command.

Parameter	Required	Description
`--alarm-name`	Yes	The name of the alarm. Must contain only UTF-8 characters.
`--comparison-operator`	Yes	The comparison operator for threshold evaluation. Valid values: `GreaterThanThreshold` , `GreaterThanOrEqualToThreshold` , `LessThanThreshold` , `LessThanOrEqualToThreshold` .
`--threshold`	Yes	The numeric threshold value to compare against.
`--query-results-to-evaluate`	Yes	The number of recent query executions to evaluate (N in M-out-of-N).
`--query-results-to-alarm`	Yes	The number of breaching results required to trigger `ALARM` (M in M-out-of-N).
`--treat-missing-data`	No	How to treat missing data. Valid values: `missing` (default), `notBreaching` , `breaching` , `ignore`.
`--scheduled-query-configuration`	Yes	The query configuration including query string, log group identifiers, scheduled query role ARN, aggregation expression, and schedule configuration.
`--action-log-line-count`	No	The number of log lines to include in Amazon SNS email notifications (0–50). Default is 0.
`--action-log-line-role-arn`	No	The ARN of the IAM role that trusts `cloudwatch.amazonaws.com` . Required if `action-log-line-count` is greater than 0.

Creating a Log Alarm using AWS CloudFormation

You can use the `AWS::CloudWatch::LogAlarm` resource type to create a Log Alarm in a AWS CloudFormation template.

The following example template creates a Log Alarm that monitors error counts.

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  HighErrorCountAlarm:
    Type: AWS::CloudWatch::LogAlarm
    Properties:
      AlarmName: HighErrorCount
      AlarmDescription: Alarm when error count exceeds 100
      ComparisonOperator: GreaterThanThreshold
      Threshold: 100
      QueryResultsToEvaluate: 5
      QueryResultsToAlarm: 3
      TreatMissingData: missing
      ActionLogLineCount: 5
      ActionLogLineRoleArn: !GetAtt LogLineRole.Arn
      ScheduledQueryConfiguration:
        QueryString: "fields @timestamp, @message | filter @message like /ERROR/"
        LogGroupIdentifiers:
          - /aws/lambda/my-function
        ScheduledQueryRoleARN: !GetAtt ScheduledQueryRole.Arn
        AggregationExpression: "count(*)"
        ScheduleConfiguration:
          ScheduleExpression: "rate(10 minutes)"
          StartTimeOffset: 600
      AlarmActions:
        - !Ref AlarmSNSTopic
```

Create a CloudWatch alarm based on a log group-metric filter

The procedure in this section describes how to create an alarm based on a log group-metric filter. With metric filters, you can look for terms and patterns in log data as the data is sent to CloudWatch. For more information, see Create metrics from log events using filters in the *Amazon CloudWatch Logs User Guide*. Before you create an alarm based on a log group-metric filter, you must complete the following actions:

- Create a log group. For more information, see Working with log groups and log streams in the *Amazon CloudWatch Logs User Guide*.
- Create a metric filter. For more information, see Create a metric filter for a log group in the *Amazon CloudWatch Logs User Guide*.

To create an alarm based on a log group-metric filter

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. From the navigation pane, choose **Logs**, and then choose **Log groups**.
3. Choose the log group that includes your metric filter.
4. Choose **Metric filters**.
5. In the metric filters tab, select the box for the metric filter that you want to base your alarm on.
6. Choose **Create alarm**.
7. (Optional) Under **Metric**, edit **Metric name**, **Statistic**, and **Period**.
8. Under **Conditions**, specify the following:
   a. For **Threshold type**, choose **Static** or **Anomaly detection**.
   b. For **Whenever *`your-metric-name`* is . . .**, choose **Greater**, **Greater/Equal**, **Lower/Equal** , or **Lower**.
   c. For **than . . .**, specify a number for your threshold value.
9. Choose **Additional configuration**.
   a. For **Data points to alarm**, specify how many data points trigger your alarm to go into the `ALARM` state. If you specify matching values, your alarm goes into the `ALARM` state if that many consecutive periods are breaching. To create an M-out-of-N alarm, specify a number for the first value that's lower than the number you specify for the second value. For more information, see Alarm evaluation.
   b. For **Missing data treatment**, select an option to specify how to treat missing data when your alarm is evaluated.
10. Choose **Next**.
11. For **Notification**, specify an Amazon SNS topic to notify when your alarm is in the `ALARM`, `OK`, or `INSUFFICIENT_DATA` state.

a. (Optional) To send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

b. (Optional) To not send notifications, choose **Remove**.

12. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

13. Choose **Next**.

14. For **Name and description**, enter a name and description for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

15. For **Preview and create**, check that your configuration is correct, and choose **Create alarm**.

Create a composite alarm

The steps in this section explain how to use the CloudWatch console to create a composite alarm. You can also use the API or AWS CLI to create a composite alarm. For more information, see PutCompositeAlarm or put-composite-alarm

> **Note**
>
> A composite alarm's rule expression can reference other alarms as children, including standard metric alarms and PromQL alarms. Composite alarms evaluate only the states of the alarms they reference. They don't evaluate metrics directly.

To create a composite alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. From the list of alarms, select the check box next to each of the existing alarms that you want to reference in your rule expression, and then choose **Create composite alarm**.
4. Under **Specify composite alarm conditions**, specify the rule expression for your new composite alarm.

   > **Note**
   >
   > Automatically, the alarms that you selected from the list of alarms are listed in the **Conditions** box. By default, the `ALARM` function has been designated to each of your alarms, and each of your alarms is joined by the logical operator `OR`.

   You can use the following substeps to modify your rule expression:

   a. You can change the required state for each of your alarms from `ALARM` to `OK` or `INSUFFICIENT_DATA`.

   b. You can change the logical operator in your rule expression from `OR` to `AND` or `NOT`, and you can add parentheses to group your functions.

   c. You can include other alarms in your rule expression or delete alarms from your rule expression.

   Example: Rule expression with conditions

```
(ALARM("CPUUtilizationTooHigh") OR
ALARM("DiskReadOpsTooHigh")) AND
```

```
OK("NetworkOutTooHigh")
```

In the example rule expression where the composite alarm goes into ALARM when ALARM ("CPUUtilizationTooHigh" or ALARM("DiskReadOpsTooHigh") is in ALARM at the same time as OK("NetworkOutTooHigh") is in OK.

5. When finished, choose **Next**.
6. Under **Configure actions**, you can choose from the following:

   For ***Notification***

   - **Select an exisiting SNS topic**, **Create a new SNS topic**, or **Use a topic ARN** to define the SNS topic that will receive the notification.
   - **Add notification**, so your alarm can send multiple notifications for the same alarm state or different alarm states.
   - **Remove** to stop your alarm from sending notifications or taking actions.

   (Optional) To have the alarm invoke a Lambda function when it changes state, choose **Add Lambda action**. Then specify the function name or ARN, and optionally choose a specific version of the function.

   For ***Systems Manager action***

   - **Add Systems Manager action**, so your alarm can perform an SSM action when it goes into ALARM.

   To learn more about Systems Manager actions, see Configuring CloudWatch to create OpsItems from alarms in the *AWS Systems Manager User Guide* and Incident creation in the *Incident Manager User Guide*. To create an alarm that performs an SSM Incident Manager action, you must have the correct permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager in the *Incident Manager User Guide*.

   To have the alarm start an investigation, choose **Add investigation action** and then select your investigation group. For more information about , see CloudWatch investigations.
7. When finished, choose **Next**.
8. Under **Add name and description**, enter an alarm name and *optional* description for your new composite alarm. The name must contain only UTF-8 characters, and can't contain ASCII

control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

9. When finished, choose **Next**.
10. Under **Preview and create**, confirm your information, and then choose **Create composite alarm**.

> **Note**
>
> You can create a cycle of composite alarms, where one composite alarm and another composite alarm depend on each other. If you find yourself in this scenario, your composite alarms stop being evaluated, and you can't delete your composite alarms because they're dependent on each other. The easiest way to break the cycle of dependecy between your composite alarms is to change the function `AlarmRule` in one of your composite alarms to `False`.

Acting on alarm changes

CloudWatch can notify users on two types of alarm changes: when an alarm changes state, and when the configuration of an alarm gets updated.

When an alarm evaluates, it might change from one state to another, such as ALARM or OK. For Metrics Insights alarms that monitor multiple time series, each time series (contributor) can only be in ALARM or OK state, never in INSUFFICIENT_DATA state. This is because a time series only exists when data is present.

Additionally, CloudWatch sends events to Amazon EventBridge whenever alarms change state, and when alarms are created, deleted, or updated. You can write EventBridge rules to take actions or be notified when EventBridge receives these events.

For more information about alarm actions, see Alarm actions.

Topics

- Notifying users on alarm changes
- Invoke a Lambda function from an alarm
- Start a CloudWatch investigations from an alarm

- Stop, terminate, reboot, or recover an EC2 instance
- Alarm events and EventBridge
- Alarm tags in action payloads

Notifying users on alarm changes

This section explains how you can use AWS User Notifications or Amazon Simple Notification Service to have users be notified of alarm changes.

Setting up AWS User Notifications

You can use AWS User Notifications to set up delivery channels to get notified about CloudWatch alarm state change and configuration change events. You receive a notification when an event matches a rule that you specify. You can receive notifications for events through multiple channels, including email, AWS Chatbot chat notifications, or AWS Console Mobile Application push notifications. You can also see notifications in the at Console Notifications Center. User Notifications supports aggregation, which can reduce the number of notifications you receive during specific events.

The notification configurations you create with AWS User Notifications do not count towards the limit on the number of actions you can configure per target alarm state. As AWS User Notifications matches the events emitted to Amazon EventBridge, it sends notifications for all the alarms in your account and selected Regions, unless you specify an advanced filter to allowlist or denylist specific alarms or patterns.

The following example of an advanced filter matches an alarm state change from OK to ALARM on the alarm named `ServerCpuTooHigh`.

```
{
"detail": {
    "alarmName": ["ServerCpuTooHigh"],
    "previousState": { "value": ["OK"] },
    "state": { "value": ["ALARM"] }
  }
}
```

You can use any of the properties published by an alarm in EventBridge events to create a filter. For more information, see Alarm events and EventBridge.

Setting up Amazon SNS notifications

You can use Amazon Simple Notification Service to send both application-to-application (A2A) messaging and application-to-person (A2P) messaging, including mobile text messaging (SMS) and email messages. For more information, see Amazon SNS event destinations.

For every state that an alarm can take, you can configure the alarm to send a message to an SNS topic. Every Amazon SNS topic you configure for a state on a given alarm counts towards the limit on the number of actions you can configure for that alarm and state. You can send messages to the same Amazon SNS topic from any alarms in your account, and use the same Amazon SNS topic for both application (A2A) and person (A2P) consumers. Because this configuration is done at the alarm level, only the alarms you have configured send messages to the selected Amazon SNS topic.

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic. For an example, see Setting up an Amazon SNS topic using the AWS Management Console. Or for more information, see Getting started with Amazon SNS.

Alternatively, if you plan to use the AWS Management Console to create your CloudWatch alarm, you can skip this procedure because you can create the topic when you create the alarm.

When you create a CloudWatch alarm, you can add actions for any target state the alarm enters. Add an Amazon SNS notification for the state you want to be notified about, and select the Amazon SNS topic you created in the previous step to send an email notification when the alarm enters the selected state.

> **Note**
>
> When you create an Amazon SNS topic, you choose to make it a *standard topic* or a *FIFO topic*. CloudWatch guarantees the publication of all alarm notifications to both types of topics. However, even if you use a FIFO topic, in rare cases CloudWatch sends the notifications to the topic out of order. If you use a FIFO topic, the alarm sets the message group ID of the alarm notifications to be a hash of the ARN of the alarm.

Topics

- Preventing confused deputy security issues
- Setting up an Amazon SNS topic using the AWS Management Console
- Setting up an SNS topic using the AWS CLI

Preventing confused deputy security issues

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn`, `aws:SourceAccount`, `aws:SourceOrgID`, and `aws:SourceOrgPaths` global condition context keys in resource policies to limit the permissions that Amazon SNS gives another service to the resource. Use `aws:SourceArn` to associate only one resource with cross-service access. Use `aws:SourceAccount` to let any resource in that account be associated with the cross-service use. Use `aws:SourceOrgID` to allow any resource from any account within an organization be associated with the cross-service use. Use `aws:SourceOrgPaths` to associate any resource from accounts within an AWS Organizations path with the cross-service use. For more information about using and understanding paths, see aws:SourceOrgPaths in the IAM User Guide.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`.

If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both `aws:SourceAccount` and `aws:SourceArn` to limit permissions.

To protect against the confused deputy problem at scale, use the `aws:SourceOrgID` or `aws:SourceOrgPaths` global condition context key with the organization ID or organization path of the resource in your resource-based policies. Policies that include the `aws:SourceOrgID` or `aws:SourceOrgPaths` key will automatically include the correct accounts and you don't have to manually update the policies when you add, remove, or move accounts in your organization.

The value of `aws:SourceArn` must be the ARN of the alarm that is sending notifications.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in CloudWatch to prevent the confused deputy problem.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "cloudwatch.amazonaws.com"
        },
        "Action": "SNS:Publish",
        "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:cloudwatch:us-east-2:111122223333:alarm:*"
            },
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            }
        }
    }]
}
```

If an alarm ARN includes any non-ASCII characters, use only the `aws:SourceAccount` global condition key to limit the permissions.

Setting up an Amazon SNS topic using the AWS Management Console

First, create a topic, then subscribe to it. You can optionally publish a test message to the topic.

To create an SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. On the Amazon SNS dashboard, under **Common actions**, choose **Create Topic**.
3. In the **Create new topic** dialog box, for **Topic name**, enter a name for the topic (for example, **`my-topic`**).
4. Choose **Create topic**.
5. Copy the **Topic ARN** for the next task (for example, arn:aws:sns:us-east-1:111122223333:my-topic).

To subscribe to an SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. In the navigation pane, choose **Subscriptions**, **Create subscription**.

3. In the **Create subscription** dialog box, for **Topic ARN**, paste the topic ARN that you created in the previous task.
4. For **Protocol**, choose **Email**.
5. For **Endpoint**, enter an email address that you can use to receive the notification, and then choose **Create subscription**.
6. From your email application, open the message from AWS Notifications and confirm your subscription.

   Your web browser displays a confirmation response from Amazon SNS.

To publish a test message to an SNS topic

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. In the navigation pane, choose **Topics**.
3. On the **Topics** page, select a topic and choose **Publish to topic**.
4. In the **Publish a message** page, for **Subject**, enter a subject line for your message, and for **Message**, enter a brief message.
5. Choose **Publish Message**.
6. Check your email to confirm that you received the message.

Setting up an SNS topic using the AWS CLI

First, you create an SNS topic, and then you publish a message directly to the topic to test that you have properly configured it.

To set up an SNS topic

1. Create the topic using the create-topic command as follows.

   ```
   aws sns create-topic --name my-topic
   ```

   Amazon SNS returns a topic ARN with the following format:

   ```
   {
       "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic"
   }
   ```

2. Subscribe your email address to the topic using the subscribe command. If the subscription request succeeds, you receive a confirmation email message.

```
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:111122223333:my-topic --
protocol email --notification-endpoint my-email-address
```

Amazon SNS returns the following:

```
{
    "SubscriptionArn": "pending confirmation"
}
```

3. From your email application, open the message from AWS Notifications and confirm your subscription.

Your web browser displays a confirmation response from Amazon Simple Notification Service.

4. Check the subscription using the list-subscriptions-by-topic command.

```
aws sns list-subscriptions-by-topic --topic-arn arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS returns the following:

```
{
  "Subscriptions": [
    {
        "Owner": "111122223333",
        "Endpoint": "me@mycompany.com",
        "Protocol": "email",
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:my-topic",
        "SubscriptionArn": "arn:aws:sns:us-east-1:111122223333:my-topic:64886986-
bf10-48fb-a2f1-dab033aa67a3"
    }
  ]
}
```

5. (Optional) Publish a test message to the topic using the publish command.

```
aws sns publish --message "Verification" --topic arn:aws:sns:us-
east-1:111122223333:my-topic
```

Amazon SNS returns the following.

```
{
    "MessageId": "42f189a0-3094-5cf6-8fd7-c2dde61a4d7d"
}
```

6. Check your email to confirm that you received the message.

Schema of Amazon SNS notifications when alarms change state

When an alarm has tag propagation enabled, these notification schemas include an `AlarmTags` list and an `AlarmTagsStatus` field. For how to enable tag propagation and what the status values mean, see Alarm tags in action payloads.

This section lists the schemas of the notifications sent to Amazon SNS topics when alarms change their state.

Schema when a metric alarm changes state

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "AlarmConfigurationUpdatedTimestamp": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OldStateValue": "string",
  "OKActions": ["string"],
  "AlarmActions": ["string"],
  "InsufficientDataActions": ["string"],
  "Trigger": {
    "MetricName": "string",
    "Namespace": "string",
    "StatisticType": "string",
    "Statistic": "string",
    "Unit": "string or null",
    "Dimensions": [
      {
```

```
        "value": "string",
        "name": "string"
      }
    ],
    "Period": "integer",
    "EvaluationPeriods": "integer",
    "DatapointsToAlarm": "integer",
    "ComparisonOperator": "string",
    "Threshold": "number",
    "TreatMissingData": "string",
    "EvaluateLowSampleCountPercentile": "string or null"
  },
  "AlarmTags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "AlarmTagsStatus": "string"
}
```

Schema when a composite alarm changes state

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OKActions": [String],
  "AlarmActions": [String],
  "InsufficientDataActions": [String],
  "OldStateValue": "string",
  "AlarmRule": "string",
  "TriggeringChildren": [String],
  "AlarmTags": [
    {
      "Key": "string",
      "Value": "string"
    }
```

```
  ],
  "AlarmTagsStatus": "string"
}
```

Schema when a log alarm changes state

```
{
  "AlarmName": "string",
  "AlarmDescription": "string",
  "AWSAccountId": "string",
  "AlarmConfigurationUpdatedTimestamp": "string",
  "NewStateValue": "string",
  "NewStateReason": "string",
  "StateChangeTime": "string",
  "Region": "string",
  "AlarmArn": "string",
  "OldStateValue": "string",
  "OKActions": ["string"],
  "AlarmActions": ["string"],
  "InsufficientDataActions": ["string"],
  "LogGroups": ["string"],
  "QueryString": "string",
  "AggregationExpression": "string",
  "QueryExecutionId": "string",
  "AlarmTags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "AlarmTagsStatus": "string"
}
```

Invoke a Lambda function from an alarm

CloudWatch alarms guarantees an asynchronous invocation of the Lambda function for a given state change, except in the following cases:

- When the function doesn't exist.
- When CloudWatch is not authorized to invoke the Lambda function.

If CloudWatch can't reach the Lambda service or the message is rejected for another reason, CloudWatch retries until the invocation is successful. Lambda queues the message and handles execution retries. For more information about this execution model, including information about how Lambda handles errors, see Asynchronous invocation in the AWS Lambda Developer Guide.

You can invoke a Lambda function in the same account, or in other AWS accounts.

When you specify an alarm to invoke a Lambda function as an alarm action, you can choose to specify the function name, function alias, or a specific version of a function.

When you specify a Lambda function as an alarm action, you must create a resource policy for the function to allow the CloudWatch service principal to invoke the function.

One way to do this is by using the AWS CLI, as in the following example:

```
aws lambda add-permission \
--function-name my-function-name \
--statement-id AlarmAction \
--action 'lambda:InvokeFunction' \
--principal lambda.alarms.cloudwatch.amazonaws.com \
--source-account 111122223333 \
--source-arn arn:aws:cloudwatch:us-east-1:111122223333:alarm:alarm-name
```

Alternatively, you can create a policy similar to one of the following examples and then assign it to the function.

The following example specifies the account where the alarm is located, so that only alarms in that account (111122223333) can invoke the function.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [{
        "Sid": "AlarmAction",
        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.alarms.cloudwatch.amazonaws.com"
        },
```

```
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-
name",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "111122223333"
            }
        }
    }]
}
```

The following example has a narrower scope, allowing only the specified alarm in the specified account to invoke the function.

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "AlarmAction",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.alarms.cloudwatch.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:444455556666:function:function-name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "111122223333",
          "AWS:SourceArn": "arn:aws:cloudwatch:us-
east-1:111122223333:alarm:alarm-name"
        }
      }
    }]
}
```

We don't recommend creating a policy that doesn't specify a source account, because such policies are vulnerable to confused deputy issues.

Add Lambda metrics to CloudWatch investigations

You can add Lambda metrics to your active CloudWatch investigations. When investigating an issue, Lambda metrics can provide valuable insights about function performance and behavior. For example, if you're investigating an application performance issue, Lambda metrics such as duration, error rates, or throttles might help identify the root cause.

To add Lambda metrics to CloudWatch investigations:

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. In the **Monitor** section, find the metric.
3. Open the context menu for the metric, choose **Investigate**, **Add to investigation**. Then, in the **Investigate** pane, select the name of the investigation.

Event object sent from CloudWatch to Lambda

When you configure a Lambda function as an alarm action, CloudWatch delivers a JSON payload to the Lambda function when it invokes the function. This JSON payload serves as the event object for the function. You can extract data from this JSON object and use it in your function. The following is an example of an event object from a metric alarm.

When the alarm has tag propagation enabled, the `alarmData` object also includes an `alarmTags` list of key-value objects and an `alarmTagsStatus` field. For information about how to enable this feature and what the status values mean, see Alarm tags in action payloads.

```
{
  'source': 'aws.cloudwatch',
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:444455556666:alarm:lambda-demo-metric-
alarm',
  'accountId': '444455556666',
  'time': '2023-08-04T12:36:15.490+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'lambda-demo-metric-alarm',
    'state': {
      'value': 'ALARM',
      'reason': 'test',
      'timestamp': '2023-08-04T12:36:15.490+0000'
    },
    'previousState': {
```

```
      'value': 'INSUFFICIENT_DATA',
      'reason': 'Insufficient Data: 5 datapoints were unknown.',
      'reasonData':
 '{"version":"1.0","queryDate":"2023-08-04T12:31:29.591+0000","statistic":"Average","period":60
[],"threshold":5.0,"evaluatedDatapoints":[{"timestamp":"2023-08-04T12:30:00.000+0000"},
{"timestamp":"2023-08-04T12:29:00.000+0000"},
{"timestamp":"2023-08-04T12:28:00.000+0000"},
{"timestamp":"2023-08-04T12:27:00.000+0000"},
{"timestamp":"2023-08-04T12:26:00.000+0000"}]}',
      'timestamp': '2023-08-04T12:31:29.595+0000'
    },
    'configuration': {
      'description': 'Metric Alarm to test Lambda actions',
      'metrics': [
        {
          'id': '1234e046-06f0-a3da-9534-EXAMPLEe4c',
          'metricStat': {
            'metric': {
              'namespace': 'AWS/Logs',
              'name': 'CallCount',
              'dimensions': {
                'InstanceId': 'i-12345678'
              }
            },
            'period': 60,
            'stat': 'Average',
            'unit': 'Percent'
          },
          'returnData': True
        }
      ]
    },
    'alarmTags': [
      { 'key': 'Environment', 'value': 'prod' }
    ],
    'alarmTagsStatus': 'SUCCESS'
  }
}
```

The following is an example of an event object from a composite alarm.

```
{
  'source': 'aws.cloudwatch',
```

```
  'alarmArn': 'arn:aws:cloudwatch:us-east-1:111122223333:alarm:SuppressionDemo.Main',
  'accountId': '111122223333',
  'time': '2023-08-04T12:56:46.138+0000',
  'region': 'us-east-1',
  'alarmData': {
    'alarmName': 'CompositeDemo.Main',
    'state': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:56:46.138+0000'
    },
    'previousState': {
      'value': 'ALARM',
      'reason': 'arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild transitioned to ALARM at Friday 04
 August, 2023 12:54:46 UTC',
      'reasonData': '{"triggeringAlarms":[{"arn":"arn:aws:cloudwatch:us-
east-1:111122223333:alarm:CompositeDemo.FirstChild","state":
{"value":"ALARM","timestamp":"2023-08-04T12:54:46.138+0000"}}]}',
      'timestamp': '2023-08-04T12:54:46.138+0000',
      'actionsSuppressedBy': 'WaitPeriod',
      'actionsSuppressedReason': 'Actions suppressed by WaitPeriod'
    },
    'configuration': {
      'alarmRule': 'ALARM(CompositeDemo.FirstChild) OR
 ALARM(CompositeDemo.SecondChild)',
      'actionsSuppressor': 'CompositeDemo.ActionsSuppressor',
      'actionsSuppressorWaitPeriod': 120,
      'actionsSuppressorExtensionPeriod': 180
    },
    'alarmTags': [
      { 'key': 'Environment', 'value': 'prod' }
    ],
    'alarmTagsStatus': 'SUCCESS'
  }
}
```

The following is an example of an event object from a log alarm.

```
{
  "source": "aws.cloudwatch",
  "alarmArn": "arn:aws:cloudwatch:us-east-1:123456789012:alarm:HighErrorRate",
  "accountId": "123456789012",
  "time": "2026-06-10T12:15:30.000+0000",
  "region": "us-east-1",
  "alarmData": {
    "alarmName": "HighErrorRate",
    "state": {
      "value": "ALARM",
      "reason": "Threshold Crossed: 3 out of the last 5 query results [142.0 (10/06/26
 12:15:00), 135.0 (10/06/26 12:10:00), 120.0 (10/06/26 12:05:00)] were greater than the
 threshold (100.0) (minimum 3 datapoints for OK -> ALARM transition).",
      "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2026-06-10T12:15:30.000+0000\",\"threshold\":100.0,\"queryResultsToEvaluate\":5,
\"queryResultsToAlarm\":3,\"results\":[...]}",
      "timestamp": "2026-06-10T12:15:30.000+0000",
      "queryExecutionId": "d70a0095-9bb5-4b9b-97f5-beced4586864"
    },
    "previousState": {
      "value": "OK",
      "reason": "...",
      "reasonData": "...",
      "timestamp": "2026-06-10T12:00:00.000+0000"
    },
    "configuration": {
      "logGroupIdentifiers": ["/aws/lambda/my-function"],
      "queryString": "fields @timestamp, @message | filter @message like /ERROR/",
      "aggregationExpression": "count(*)",
      "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/ScheduledQueryRole",
      "schedule": {
        "expression": "rate(5 minutes)",
        "startTimeOffset": 300,
        "endTimeOffset": 0
      },
      "threshold": 100.0,
      "comparisonOperator": "GreaterThanThreshold",
      "treatMissingData": "missing",
      "queryResultsToEvaluate": 5,
      "queryResultsToAlarm": 3,
      "alarmName": "HighErrorRate",
      "description": "Alarm when error count exceeds 100",
      "actionsEnabled": true,
```

```
      "timestamp": "2026-06-10T12:00:00.000+0000",
      "okActions": [],
      "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
      "insufficientDataActions": []
    },
    "alarmTags": [
      { "key": "Environment", "value": "prod" }
    ],
    "alarmTagsStatus": "SUCCESS"
  }
}
```

Start a CloudWatch investigations from an alarm

Start a CloudWatch investigations from an alarm, or from any point in the last two weeks of a CloudWatch alarm's history.

For more information about CloudWatch investigations, see CloudWatch investigations.

Prerequisites

Before you can start a CloudWatch investigations from a CloudWatch alarm, you must create a resource policy for the function to allow the CloudWatch service principal to start the investigation. To do this using the AWS CLI, use a command similar to the following example:

```
aws aiops put-investigation-group-policy \
    --identifier arn:aws:aiops:us-east-1:111122223333:investigation-group/
investigation_group_id \
    --policy "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":\"Allow\",
\"Principal\":{\"Service\":\"aiops.alarms.cloudwatch.amazonaws.com\"},\"Action\":
[\"aiops:CreateInvestigation\",\"aiops:CreateInvestigationEvent\"],\"Resource\":\"*\",
\"Condition\":{\"StringEquals\":{\"aws:SourceAccount\":\"111122223333\"},\"ArnLike\":
{\"aws:SourceArn\":\"arn:aws:cloudwatch:us-east-1:111122223333:alarm:*\"}}}]}" \
    --region eu-north-1
```

Replace the example values with your own AWS account ID, region, and investigation group ID.

Start an investigation from a CloudWatch alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the left navigation pane, choose **Alarms**, **All alarms**.
3. Choose the name of the alarm.
4. Choose the time period in the alarm history that you want to investigate.
5. Choose **Investigate**, **Start new investigation**.
6. For **New investigation title**, enter a name for the investigation. Then choose **Start investigation**.

   The CloudWatch investigations assistant starts and scans your telemetry data to find data that might be associated with this situation.
7. In the CloudWatch console's navigation pane, choose **Investigations**, then choose the name of the investigation that you just started.

   The **Findings** section displays a natural-language summary of the alarm's status and the reason that it was triggered.
8. (Optional) In the graph of the alarm, right-click and choose to deep-dive into the alarm or the metric that it watches.
9. On the right side of the screen, choose the **Suggestions** tab.

   A list of other telemetry that CloudWatch investigations has discovered and that might be relevant to the investigation appears. These findings can include other metrics and CloudWatch Logs Insights query results. CloudWatch investigations ran these queries based on the alarm.

   - For each finding, choose **Add to findings** or **Discard**.

     When you choose **Add to findings**, the telemetry is added to the **Findings** section, and CloudWatch investigations uses this information to direct its further scanning and suggestions.
   - For a CloudWatch Logs Insights query result, to change or edit the query and re-run it, open the context (right-click) menu for the results, and then choose **Open in Logs Insights**. For more information, see Analyzing log data with CloudWatch Logs Insights.

     To run a different query, when you get to the Logs Insights page, choose to use query assist to form a query using natural language. For more information, see Use natural language to generate and update CloudWatch Logs Insights queries.
   - (Optional) If you know of telemetry in another AWS service that might apply to this investigation, go to that service's console and add the telemetry to the investigation.

10. CloudWatch investigations might also add hypotheses to the list in the **Suggestions** tab. These hypotheses are generated by the investigation in natural language.

    For each hypothesis, choose **Add to findings** or **Discard**.
11. When you think you have completed the investigation and found the root cause of the issue, choose the **Overview** tab and then choose **Investigation summary**. CloudWatch investigations then creates a natural-language summary of the important findings and hypotheses from the investigation.

Stop, terminate, reboot, or recover an EC2 instance

Using Amazon CloudWatch alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your EC2 instances. You can use the stop or terminate actions to help you save money when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances or recover them onto new hardware if a system impairment occurs.

There are a number of scenarios in which you might want to automatically stop or terminate your instance. For example, you might have instances dedicated to batch payroll processing jobs or scientific computing tasks that run for a period of time and then complete their work. Rather than letting those instances sit idle (and accrue charges), you can stop or terminate them, which helps you to save money. The main difference between using the stop and the terminate alarm actions is that you can easily restart a stopped instance if you need to run it again later. You can also keep the same instance ID and root volume. However, you cannot restart a terminated instance. Instead, you must launch a new instance.

You can add the stop, terminate, or reboot, actions to any alarm that is set on an Amazon EC2 per-instance metric, including basic and detailed monitoring metrics provided by Amazon CloudWatch (in the AWS/EC2 namespace), in addition to any custom metrics that include the "InstanceId=" dimension, as long as the InstanceId value refers to a valid running Amazon EC2 instance. You can also add the recover action to alarms that is set on any Amazon EC2 per-instance metric except for `StatusCheckFailed_Instance`.

> **Important**
>
> Alarms configured on Amazon EC2 metrics can temporarily enter the INSUFFICIENT_DATA state if there are missing metric data points. This is rare, but can happen when the metric reporting is interrupted, even when the Amazon EC2 instance is healthy. For alarms on

Amazon EC2 metrics that are configured to take stop, terminate, reboot, or recover actions, we recommend that you configure those alarms to treat missing data as `missing`, and to have these alarms trigger only when in the ALARM state.
For more information about how you can configure CloudWatch to act on missing metrics that have alarms set on them, see Configuring how CloudWatch alarms treat missing data.

To set up a CloudWatch alarm action that can reboot, stop, or terminate an instance, you must use a service-linked IAM role, *AWSServiceRoleForCloudWatchEvents*. The AWSServiceRoleForCloudWatchEvents IAM role enables AWS to perform alarm actions on your behalf.

To create the service-linked role for CloudWatch Events, use the following command:

```
aws iam create-service-linked-role --aws-service-name events.amazonaws.com
```

Console support

You can create alarms using the CloudWatch console or the Amazon EC2 console. The procedures in this documentation use the CloudWatch console. For procedures that use the Amazon EC2 console, see Create alarms that stop, terminate, reboot, or rcover an instance in the *Amazon EC2 User Guide*.

Permissions

If you are using an AWS Identity and Access Management (IAM) account to create or modify an alarm that performs EC2 actions or Systems Manager OpsItem actions, you must have the `iam:CreateServiceLinkedRole` permission.

Contents

- Adding stop actions to Amazon CloudWatch alarms
- Adding terminate actions to Amazon CloudWatch alarms
- Adding reboot actions to Amazon CloudWatch alarms
- Adding recover actions to Amazon CloudWatch alarms
- Viewing the history of triggered alarms and actions

Adding stop actions to Amazon CloudWatch alarms

You can create an alarm that stops an Amazon EC2 instance when a certain threshold has been met. For example, you may run development or test instances and occasionally forget to shut them off. You can create an alarm that is triggered when the average CPU utilization percentage has been lower than 10 percent for 24 hours, signaling that it is idle and no longer in use. You can adjust the threshold, duration, and period to suit your needs, plus you can add an SNS notification, so that you will receive an email when the alarm is triggered.

Amazon EC2 instances that use an Amazon Elastic Block Store volume as the root device can be stopped or terminated, whereas instances that use the instance store as the root device can only be terminated.

To create an alarm to stop an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. For **AWS namespaces**, choose **EC2**.
6. Do the following:
    a. Choose **Per-Instance Metrics**.
    b. Select the check box in the row with the correct instance and the **CPUUtilization** metric.
    c. Choose the **Graphed metrics** tab.
    d. For the statistic, choose **Average**.
    e. Choose a period (for example, **1 Hour**).
    f. Choose **Select metric**.
7. Under **Conditions**, do the following:
    a. Choose **Static**.
    b. Under **Whenever CPUUtilization is**, choose **Lower**.
    c. For **than**, type **10**.
    d. Choose **Next**.
    e. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Stop_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

f. Choose **Add EC2 Action**.

g. For **Alarm state trigger**, choose **In alarm**. For **Take the following action**, choose **Stop this instance**.

h. Choose **Next**.

i. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

j. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Adding terminate actions to Amazon CloudWatch alarms

You can create an alarm that terminates an EC2 instance automatically when a certain threshold has been met (as long as termination protection is not enabled for the instance). For example, you might want to terminate an instance when it has completed its work, and you don't need the instance again. If you might want to use the instance later, you should stop the instance instead of terminating it. For information about enabling and disabling termination protection for an instance, see Enabling Termination Protection for an Instance in the *Amazon EC2 User Guide*.

To create an alarm to terminate an idle instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. For the **Select Metric** step, do the following:

   a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.

   b. Select the row with the instance and the **CPUUtilization** metric.

   c. For the statistic, choose **Average**.

   d. Choose a period (for example, **1 Hour**).

   e. Choose **Next**.
4. For the **Define Alarm** step, do the following:

a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Terminate EC2 instance) and a description of the alarm (for example, Terminate EC2 instance when CPU is idle for too long). Alarm names must contain only ASCII characters.

b. Under **Whenever**, for **is**, choose **<** and type **10**. For **for**, type **24** consecutive periods.

   A graphical representation of the threshold is shown under **Alarm Preview**.

c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

   To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Terminate_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

d. Choose **EC2 Action**.

e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Terminate this instance**.

f. Choose **Create Alarm**.

Adding reboot actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically reboots the instance. The reboot alarm action is recommended for Instance Health Check failures (as opposed to the recover alarm action, which is suited for System Health Check failures). An instance reboot is equivalent to an operating system reboot. In most cases, it takes only a few minutes to reboot your instance. When you reboot an instance, it remains on the same physical host, so your instance keeps its public DNS name, private IP address, and any data on its instance store volumes.

Rebooting an instance doesn't start a new instance billing hour, unlike stopping and restarting your instance. For more information about rebooting an instance, see Reboot Your Instance in the *Amazon EC2 User Guide*.

> **Important**
>
> To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set reboot alarms to three evaluation periods of one minute each.

To create an alarm to reboot an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **Create Alarm**.
3. For the **Select Metric** step, do the following:
   a. Under **EC2 Metrics**, choose **Per-Instance Metrics**.
   b. Select the row with the instance and the **StatusCheckFailed_Instance** metric.
   c. For the statistic, choose **Minimum**.
   d. Choose a period (for example, **`1 Minute`**).
   e. Choose **Next**.
4. For the **Define Alarm** step, do the following:
   a. Under **Alarm Threshold**, type a unique name for the alarm (for example, Reboot EC2 instance) and a description of the alarm (for example, Reboot EC2 instance when health checks fail). Alarm names must contain only ASCII characters.
   b. Under **Whenever**, for **is**, choose **>** and type **`0`**. For **for**, type **3** consecutive periods.

      A graphical representation of the threshold is shown under **Alarm Preview**.
   c. Under **Notification**, for **Send notification to**, choose an existing SNS topic or create a new one.

      To create an SNS topic, choose **New list**. For **Send notification to**, type a name for the SNS topic (for example, Reboot_EC2_Instance). For **Email list**, type a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.
   d. Choose **EC2 Action**.

e. For **Whenever this alarm**, choose **State is ALARM**. For **Take this action**, choose **Reboot this instance**.

f. Choose **Create Alarm**.

Adding recover actions to Amazon CloudWatch alarms

You can create an Amazon CloudWatch alarm that monitors an Amazon EC2 instance and automatically recovers the instance if it becomes impaired due to an underlying hardware failure or a problem that requires AWS involvement to repair. Terminated instances cannot be recovered. A recovered instance is identical to the original instance, including the instance ID, private IP addresses, Elastic IP addresses, and all instance metadata.

When the `StatusCheckFailed_System` alarm is triggered, and the recover action is initiated, you will be notified by the Amazon SNS topic that you chose when you created the alarm and associated the recover action. During instance recovery, the instance is migrated during an instance reboot, and any data that is in-memory is lost. When the process is complete, information is published to the SNS topic you've configured for the alarm. Anyone who is subscribed to this SNS topic will receive an email notification that includes the status of the recovery attempt and any further instructions. You will notice an instance reboot on the recovered instance.

The recover action can be used only with `StatusCheckFailed_System`, not with `StatusCheckFailed_Instance`.

Examples of problems that cause system status checks to fail include:

- Loss of network connectivity
- Loss of system power
- Software issues on the physical host
- Hardware issues on the physical host that impact network reachability

The recover action is supported only on some instance types. For more information about supported instance types and other requirements, see Recover your instance and Requirements.

> **Important**
>
> To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set

recover alarms to two evaluation periods of one minute each and reboot alarms to three evaluation periods of one minute each.

To create an alarm to recover an instance using the Amazon CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create Alarm**.
4. Choose **Select Metric** and then do the following:
   a. Choose **EC2 Metrics**, **Per-Instance Metrics**.
   b. Select the row with the instance and the **StatusCheckFailed_System** metric, and then choose **Select metric**.
   c. For the statistic, choose **Minimum**.
   d. Choose a period (for example, **`1 Minute`**).

   > **Important**
   >
   > To avoid a race condition between the reboot and recover actions, avoid setting the same evaluation period for both a reboot alarm and a recover alarm. We recommend that you set recover alarms to two evaluation periods of one minute each.

5. For **Conditions**, do the following:
   a. Under **Threshold type**, choose **Static**.
   b. Under **Whenever**, choose **Greater** and enter **`0`** for **than...**.
   c. Choose **Additional configuration**, then for **Datapoints to alarm** specify 2 **out of** 2.
6. Choose **Next**.
7. Under **Notification**, do the following:
   a. For **Alarm state trigger**, choose **In alarm**.
   b. For **Send notification to the following SNS topic**, choose an existing SNS topic or create a new one.
   c. Choose **Add EC2 Action**.

d. For **Alarm state trigger**, choose **In alarm**.

e. For **Take the following action**, choose **Recover this instance**.

f. Choose **Next**.

8. For **Alarm name**, type a unique name for the alarm (for example, **`Recover EC2 instance`**) and a description of the alarm (for example, **`Recover EC2 instance when health checks fail`**). Alarm names must contain only ASCII characters.
9. Choose **Next**.
10. Choose **Create Alarm**.

Viewing the history of triggered alarms and actions

You can view alarm and action history in the Amazon CloudWatch console. Amazon CloudWatch keeps the last 30 days of alarm and action history.

To view the history of triggered alarms and actions

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms** and select an alarm.
3. To view the most recent state transition along with the time and metric values, choose **Details**.
4. To view the most recent history entries, choose **History**.

Alarm events and EventBridge

CloudWatch sends events to Amazon EventBridge whenever a CloudWatch alarm is created, updated, deleted, or changes alarm state. You can use EventBridge and these events to write rules that take actions, such as notifying you, when an alarm changes state. For more information, see What is Amazon EventBridge?

CloudWatch guarantees the delivery of alarm state change events to EventBridge.

Alarm State Change Events

This section shows example events sent to EventBridge when an alarm's state changes. Select a tab to view different types of alarm state change events.

When an alarm has tag propagation enabled, the event `detail` also includes an `alarmTags` object (a set of key-value pairs, which makes event pattern filtering easier) and an

`alarmTagsStatus` field. For how to enable tag propagation and what the status values mean, see Alarm tags in action payloads.

Single Metric Alarm

Events generated when a single metric alarm changes state. These events include `state` and `previousState` fields with the alarm's evaluation results.

```
{
    "version": "0",
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:04:40Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "configuration": {
            "description": "Goes into alarm when server CPU utilization is too
 high!",
            "metrics": [
                {
                    "id": "30b6c6b2-a864-43a2-4877-c09a1afc3b87",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "CPUUtilization",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ]
        },
        "previousState": {
```

```
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [0.0666851903306472 (01/10/19 13:46:00)] was not greater than the threshold (50.0)
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-01T13:56:40.985+0000\",\"startDate\":\"2019-10-01T13:46:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[0.0666851903306472],
\"threshold\":50.0}",
            "timestamp": "2019-10-01T13:56:40.987+0000",
            "value": "OK"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints
 [99.50160229693434 (02/10/19 16:59:00)] was greater than the threshold (50.0)
 (minimum 1 datapoint for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:04:40.985+0000\",\"startDate\":\"2019-10-02T16:59:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[99.50160229693434],
\"threshold\":50.0}",
            "timestamp": "2019-10-02T17:04:40.989+0000",
            "value": "ALARM"
        },
        "muteDetail": {
            "mutedByArn": "arn:aws:cloudwatch:us-east-1:1234567890:alarm-mute-
rule:testMute",
            "muteWindowStart": "2026-01-01T10:00:00.000+0000",
            "muteWindowEnd": "2026-01-01T12:00:00.000+0000"
        },
        "alarmTags": {
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
    }
}
```

Metric Math Alarm

Events generated when a metric math alarm changes state. These events include the math expression details in the `configuration` field.

```
{
    "version": "0",
    "id": "2dde0eb1-528b-d2d5-9ca6-6d590caf2329",
```

```
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-02T17:20:48Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "configuration": {
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "metrics": [
                {
                    "expression": "SUM(METRICS())",
                    "id": "e1",
                    "label": "Total Network Traffic",
                    "returnData": true
                },
                {
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkIn",
                            "namespace": "AWS/EC2"
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            },
                            "name": "NetworkOut",
                            "namespace": "AWS/EC2"
                        },
```

```
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                }
            ]
        },
        "previousState": {
            "reason": "Unchecked: Initial alarm creation",
            "timestamp": "2019-10-02T17:20:03.642+0000",
            "value": "INSUFFICIENT_DATA"
        },
        "state": {
            "reason": "Threshold Crossed: 1 out of the last 1 datapoints [45628.0
 (02/10/19 17:10:00)] was greater than the threshold (10000.0) (minimum 1 datapoint
 for OK -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-02T17:20:48.551+0000\",\"startDate\":\"2019-10-02T17:10:00.000+0000\",
\"period\":300,\"recentDatapoints\":[45628.0],\"threshold\":10000.0}",
            "timestamp": "2019-10-02T17:20:48.554+0000",
            "value": "ALARM"
        },
        "muteDetail": {
            "mutedByArn": "arn:aws:cloudwatch:us-east-1:1234567890:alarm-mute-
rule:testMute",
            "muteWindowStart": "2026-01-01T10:00:00.000+0000",
            "muteWindowEnd": "2026-01-01T12:00:00.000+0000"
        },
        "alarmTags": {
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
    }
}
```

Anomaly Detection Alarm

Events generated when an anomaly detection alarm changes state. These events include upper and lower threshold bounds in the `reasonData` field.

```
{
    "version": "0",
```

```
    "id": "daafc9f1-bddd-c6c9-83af-74971fcfc4ef",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2019-10-03T16:00:04Z",
    "region": "us-east-1",
    "resources": ["arn:aws:cloudwatch:us-east-1:123456789012:alarm:EC2 CPU
 Utilization Anomaly"],
    "detail": {
        "alarmName": "EC2 CPU Utilization Anomaly",
        "state": {
            "value": "ALARM",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints [0.0
 (03/10/19 15:58:00)] was less than the lower thresholds [0.020599444741798756] or
 greater than the upper thresholds [0.3006915352732461] (minimum 1 datapoint for OK
 -> ALARM transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T16:00:04.650+0000\",\"startDate\":\"2019-10-03T15:58:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.0],\"recentLowerThresholds\":
[0.020599444741798756],\"recentUpperThresholds\":[0.3006915352732461]}",
            "timestamp": "2019-10-03T16:00:04.653+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "Thresholds Crossed: 1 out of the last 1 datapoints
 [0.166666666664241 (03/10/19 15:57:00)] was not less than the lower thresholds
 [0.0206719426210418] or not greater than the upper thresholds [0.30076870222143803]
 (minimum 1 datapoint for ALARM -> OK transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2019-10-03T15:59:04.670+0000\",\"startDate\":\"2019-10-03T15:57:00.000+0000\",
\"period\":60,\"recentDatapoints\":[0.166666666664241],\"recentLowerThresholds\":
[0.0206719426210418],\"recentUpperThresholds\":[0.30076870222143803]}",
            "timestamp": "2019-10-03T15:59:04.672+0000"
        },
        "muteDetail": {
            "mutedByArn": "arn:aws:cloudwatch:us-east-1:1234567890:alarm-mute-
rule:testMute",
            "muteWindowStart": "2026-01-01T10:00:00.000+0000",
            "muteWindowEnd": "2026-01-01T12:00:00.000+0000"
        },
        "configuration": {
            "description": "Goes into alarm if CPU Utilization is out of band",
            "metrics": [{
                "id": "m1",
```

```
                "metricStat": {
                    "metric": {
                        "namespace": "AWS/EC2",
                        "name": "CPUUtilization",
                        "dimensions": {
                            "InstanceId": "i-12345678901234567"
                        }
                    },
                    "period": 60,
                    "stat": "Average"
                },
                "returnData": true
            }, {
                "id": "ad1",
                "expression": "ANOMALY_DETECTION_BAND(m1, 0.8)",
                "label": "CPUUtilization (expected)",
                "returnData": true
            }]
        },
        "alarmTags": {
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
    }
}
```

Composite Alarm

Events generated when a composite alarm changes state. These events include suppression information in the `actionsSuppressedBy` and `actionsSuppressedReason` fields.

```
{
    "version": "0",
    "id": "d3dfc86d-384d-24c8-0345-9f7986db0b80",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-22T15:57:45Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
```

```
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "actionsSuppressedReason": "Actions suppressed by WaitPeriod",
            "value": "ALARM",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.FirstChild transitioned to
 ALARM at Friday 22 July, 2022 15:57:45 UTC",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"ALARM\",
\"timestamp\":\"2022-07-22T15:57:45.394+0000\"}}]}",
            "timestamp": "2022-07-22T15:57:45.394+0000"
        },
        "previousState": {
            "value": "OK",
            "reason": "arn:aws:cloudwatch:us-
east-1:123456789012:alarm:SuppressionDemo.EventBridge.Main was created and its alarm
 rule evaluates to OK",
            "reasonData": "{\"triggeringAlarms\":[{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:TotalNetworkTrafficTooHigh\",\"state\":{\"value\":\"OK\",
\"timestamp\":\"2022-07-14T16:28:57.770+0000\"}},{\"arn\":\"arn:aws:cloudwatch:us-
east-1:123456789012:alarm:ServerCpuTooHigh\",\"state\":{\"value\":\"OK\",\"timestamp
\":\"2022-07-14T16:28:54.191+0000\"}}]}",
            "timestamp": "2022-07-22T15:56:14.552+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180
        },
        "muteDetail": {
            "mutedByArn": "arn:aws:cloudwatch:us-east-1:1234567890:alarm-mute-
rule:testMute",
            "muteWindowStart": "2026-01-01T10:00:00.000+0000",
            "muteWindowEnd": "2026-01-01T12:00:00.000+0000"
        },
        "alarmTags": {
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
```

```
    }
}
```

Multi Time Series Alarm

Events generated when an Alarm Contributor or an Alarm changes state. Alarm Contributor state change events contain the id and attributes of the Alarm Contributor as well as the most recent datapoint that breached the threshold. Alarm State change events have a summary of the number of contributors that caused the alarm to transition in their state reason.

Alarm Contributor Example

```
{
  "version": "0",
  "id": "6d226bbc-07f0-9a31-3359-1736968f8ded",
  "detail-type": "CloudWatch Alarm Contributor State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2025-12-01T13:42:04Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:DynamoDBInsightsAlarm"
  ],
  "detail": {
    "alarmName": "DynamoDBInsightsAlarm",
    "alarmContributor": {
      "id": "6d442278dba546f6",
      "attributes": {
        "TableName": "example-dynamodb-table-name"
      }
    },
    "state": {
      "value": "ALARM",
      "reason": "Threshold Crossed: 1 datapoint was less than the threshold (1.0).
 The most recent datapoint which crossed the threshold: [0.0 (01/12/25 13:34:00)].",
      "timestamp": "2025-12-01T13:42:04.919+0000"
    },
    "configuration": {
      "metrics": [
        {
          "id": "m1",
          "expression": "SELECT AVG(ConsumedWriteCapacityUnits) FROM \"AWS/DynamoDB
\" GROUP BY TableName ORDER BY MAX() DESC",
```

```
          "returnData":true,
          "period": 60
        }
      ],
      "description": "Metrics Insights alarm for DynamoDB ConsumedWriteCapacity per
 TableName"
    },
    "muteDetail": {
        "mutedByArn": "arn:aws:cloudwatch:us-east-1:1234567890:alarm-mute-
rule:testMute",
        "muteWindowStart": "2026-01-01T10:00:00.000+0000",
        "muteWindowEnd": "2026-01-01T12:00:00.000+0000"
    },
    "alarmTags": {
      "Environment": "prod",
      "Team": "cloudwatch"
    },
    "alarmTagsStatus": "SUCCESS"
  }
}
```

Alarm Example

```
{
  "version": "0",
  "id": "80ddd249-dedf-7c4d-0708-0eb78132dd78",
  "detail-type": "CloudWatch Alarm State Change",
  "source": "aws.cloudwatch",
  "account": "123456789012",
  "time": "2025-12-01T13:42:04Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:cloudwatch:us-east-1:123456789012:alarm:DynamoDBInsightsAlarm"
  ],
  "detail": {
    "alarmName": "DynamoDBInsightsAlarm",
    "state": {
      "value": "ALARM",
      "reason": "6 out of 6 time series evaluated to ALARM",
      "timestamp": "2025-12-01T13:42:04.919+0000"
    },
    "previousState": {
      "value": "INSUFFICIENT_DATA",
```

```
      "reason": "Unchecked: Initial alarm creation",
      "timestamp": "2025-12-01T13:40:50.600+0000"
    },
    "configuration": {
      "metrics": [
        {
          "id": "m1",
          "expression": "SELECT AVG(ConsumedWriteCapacityUnits) FROM \"AWS/DynamoDB
\" GROUP BY TableName ORDER BY MAX() DESC",
          "returnData": true,
          "period": 60
        }
      ],
      "description": "Metrics Insights alarm for DynamoDB ConsumedWriteCapacity per
 TableName"
    },
    "alarmTags": {
      "Environment": "prod",
      "Team": "cloudwatch"
    },
    "alarmTagsStatus": "SUCCESS"
  }
}
```

Log Alarm

Events generated when a single-value log alarm (without a BY clause) changes state.

```
{
    "version": "0",
    "id": "c4c1c1c9-6542-e61b-6ef0-8c4d36933a92",
    "detail-type": "CloudWatch Alarm State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2026-06-10T12:15:30Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:HighErrorRate"
    ],
    "detail": {
        "alarmName": "HighErrorRate",
        "state": {
            "value": "ALARM",
```

```
            "reason": "Threshold Crossed: 3 out of the last 5 query results
 [142.0 (10/06/26 12:15:00), 135.0 (10/06/26 12:10:00), 120.0 (10/06/26 12:05:00)]
 were greater than the threshold (100.0) (minimum 3 datapoints for OK -> ALARM
 transition).",
            "reasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2026-06-10T12:15:30.000+0000\",\"threshold\":100.0,\"queryResultsToEvaluate\":5,
\"queryResultsToAlarm\":3,\"results\":[...]}",
            "timestamp": "2026-06-10T12:15:30.000+0000",
            "queryExecutionId": "d70a0095-9bb5-4b9b-97f5-beced4586864"
        },
        "previousState": {
            "value": "OK",
            "reason": "...",
            "reasonData": "...",
            "timestamp": "2026-06-10T12:00:00.000+0000"
        },
        "configuration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function"],
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*)",
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 0
            },
            "threshold": 10.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "notBreaching",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "HighErrorRate",
            "description": "Alarm when error count exceeds threshold",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T12:00:00.000+0000",
            "okActions": [],
            "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
            "insufficientDataActions": []
        },
        "alarmTags": {
```

```
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
    }
}
```

Multi-Contributor Log Alarm

Events generated when a contributor of a multi-contributor log alarm (with a BY clause) changes state. Each contributor generates a separate event.

```
{
    "version": "0",
    "id": "6d226bbc-07f0-9a31-3359-1736968f8ded",
    "detail-type": "CloudWatch Alarm Contributor State Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2026-06-10T12:20:15Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:EndpointLatency"
    ],
    "detail": {
        "alarmName": "EndpointLatency",
        "alarmContributor": {
            "id": "a1b2c3d4e5f6g7h8",
            "attributes": {
                "endpoint": "/api/orders"
            }
        },
        "state": {
            "value": "ALARM",
            "reason": "Threshold Crossed: 3 out of the last 5 query results [1250.0
 (10/06/26 12:20:00), 1100.0 (10/06/26 12:15:00), 1050.0 (10/06/26 12:10:00)]
 were greater than the threshold (1000.0) (minimum 3 datapoints for OK -> ALARM
 transition).",
            "timestamp": "2026-06-10T12:20:15.000+0000",
            "queryExecutionId": "d70a0095-9bb5-4b9b-97f5-beced4586864"
        },
        "configuration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function", "/aws/ecs/my-
service"],
```

```
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*) by serviceName",
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 60
            },
            "threshold": 10.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "notBreaching",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "EndpointLatency",
            "description": "description",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T12:00:00.000+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        },
        "alarmTags": {
            "Environment": "prod",
            "Team": "cloudwatch"
        },
        "alarmTagsStatus": "SUCCESS"
    }
}
```


Alarm Configuration Change Events

This section shows example events sent to EventBridge when an alarm's configuration changes. Configuration changes include creating, updating, or deleting alarms.

Creation Events

Events generated when new alarms are created. These events include the initial alarm configuration in the `configuration` field, with `operation` set to "create".

Composite Alarm Example

```
{
    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:22Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:22.289+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "alarmName": "ServiceAggregatedAlarm",
            "description": "Aggregated monitor for instance",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:22.289+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Composite Alarm with Suppressor Example

```
{
    "version": "0",
    "id": "454773e1-09f7-945b-aa2c-590af1c3f8e0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:46Z",
```

```
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "create",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Log Alarm Example

```
{
    "version": "0",
    "id": "91535fdd-1e9c-849d-624b-9a9f2b1d09d0",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2026-06-10T12:00:00Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:HighErrorRate"
    ],
    "detail": {
        "alarmName": "HighErrorRate",
        "operation": "create",
```

```
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2026-06-10T12:00:00.000+0000"
        },
        "configuration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function"],
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*)",
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 0
            },
            "threshold": 100.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "missing",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "HighErrorRate",
            "description": "Alarm when error count exceeds 100",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T12:00:00.000+0000",
            "okActions": [],
            "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
            "insufficientDataActions": []
        }
    }
}
```

Update Events

Events generated when existing alarms are modified. These events contain both `configuration` and `previousConfiguration` fields to show what changed.

Metric Alarm Example

```
{
    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
```

```
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:06:34Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServerCpuTooHigh"
    ],
    "detail": {
        "alarmName": "ServerCpuTooHigh",
        "operation": "update",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:13.757+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 80,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "86bfa85f-b14c-ebf7-8916-7da014ce23c0",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "CPUUtilization",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ],
            "alarmName": "ServerCpuTooHigh",
            "description": "Goes into alarm when server CPU utilization is too
high!",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:34.267+0000",
            "okActions": [],
            "alarmActions": [],
```

```
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "evaluationPeriods": 1,
            "threshold": 70,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [
                {
                    "id": "d6bfa85f-893e-b052-a58b-4f9295c9111a",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "CPUUtilization",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Average"
                    },
                    "returnData": true
                }
            ],
            "alarmName": "ServerCpuTooHigh",
            "description": "Goes into alarm when server CPU utilization is too
 high!",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:13.757+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Metric Alarm with Suppressor Example

```
    {
    "version": "0",
    "id": "4c6f4177-6bd5-c0ca-9f05-b4151c54568b",
    "detail-type": "CloudWatch Alarm Configuration Change",
```

```
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T13:59:56Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "update",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [], Remove
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 180,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:46.425+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Log Alarm Update Example


```
{
    "version": "0",
    "id": "bc7d3391-47f8-ae47-f457-1b4d06118d50",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2026-06-10T13:00:00Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:HighErrorRate"
    ],
    "detail": {
        "alarmName": "HighErrorRate",
        "operation": "update",
        "state": {
            "value": "ALARM",
            "timestamp": "2026-06-10T12:15:30.000+0000"
        },
        "configuration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function"],
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*)",
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 0
            },
            "threshold": 200.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "missing",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "HighErrorRate",
            "description": "Updated description",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T13:00:00.000+0000",
            "okActions": [],
```

```
            "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
            "insufficientDataActions": []
        },
        "previousConfiguration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function"],
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*)",
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 0
            },
            "threshold": 100.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "missing",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "HighErrorRate",
            "description": "Alarm when error count exceeds 100",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T12:00:00.000+0000",
            "okActions": [],
            "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
            "insufficientDataActions": []
        }
    }
}
```

Deletion Events

Events generated when alarms are deleted. These events include the final alarm configuration and set `operation` to "delete".

Metric Math Alarm Example

```
{
    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
```

```
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-03-03T17:07:13Z",
    "region": "us-east-*",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:TotalNetworkTrafficTooHigh"
    ],
    "detail": {
        "alarmName": "TotalNetworkTrafficTooHigh",
        "operation": "delete",
        "state": {
            "value": "INSUFFICIENT_DATA",
            "timestamp": "2022-03-03T17:06:17.672+0000"
        },
        "configuration": {
            "evaluationPeriods": 1,
            "threshold": 10000,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "ignore",
            "metrics": [{
                    "id": "m1",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkIn",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "m2",
                    "metricStat": {
                        "metric": {
                            "namespace": "AWS/EC2",
                            "name": "NetworkOut",
                            "dimensions": {
                                "InstanceId": "i-12345678901234567"
                            }
                        },
```

```
                        "period": 300,
                        "stat": "Maximum"
                    },
                    "returnData": false
                },
                {
                    "id": "e1",
                    "expression": "SUM(METRICS())",
                    "label": "Total Network Traffic",
                    "returnData": true
                }
            ],
            "alarmName": "TotalNetworkTrafficTooHigh",
            "description": "Goes into alarm if total network traffic exceeds 10Kb",
            "actionsEnabled": true,
            "timestamp": "2022-03-03T17:06:17.672+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Metric Math Alarm with Suppressor Example

```
{
    "version": "0",
    "id": "e34592a1-46c0-b316-f614-1b17a87be9dc",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2022-07-14T14:00:01Z",
    "region": "us-east-*",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:ServiceAggregatedAlarm"
    ],
    "detail": {
        "alarmName": "ServiceAggregatedAlarm",
        "operation": "delete",
        "state": {
            "actionsSuppressedBy": "WaitPeriod",
            "value": "ALARM",
            "timestamp": "2022-07-14T13:59:46.425+0000"
```

```
        },
        "configuration": {
            "alarmRule": "ALARM(ServerCpuTooHigh) OR
 ALARM(TotalNetworkTrafficTooHigh)",
            "actionsSuppressor": "ServiceMaintenanceAlarm",
            "actionsSuppressorWaitPeriod": 120,
            "actionsSuppressorExtensionPeriod": 360,
            "alarmName": "ServiceAggregatedAlarm",
            "actionsEnabled": true,
            "timestamp": "2022-07-14T13:59:56.290+0000",
            "okActions": [],
            "alarmActions": [],
            "insufficientDataActions": []
        }
    }
}
```

Log Alarm Deletion Example

```
{
    "version": "0",
    "id": "f171d220-9e1c-c252-5042-2677347a83ed",
    "detail-type": "CloudWatch Alarm Configuration Change",
    "source": "aws.cloudwatch",
    "account": "123456789012",
    "time": "2026-06-10T14:00:00Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:cloudwatch:us-east-1:123456789012:alarm:HighErrorRate"
    ],
    "detail": {
        "alarmName": "HighErrorRate",
        "operation": "delete",
        "state": {
            "value": "ALARM",
            "timestamp": "2026-06-10T12:15:30.000+0000"
        },
        "configuration": {
            "logGroupIdentifiers": ["/aws/lambda/my-function"],
            "queryString": "fields @timestamp, @message | filter @message like /
ERROR/",
            "aggregationExpression": "count(*)",
```

```
            "scheduledQueryRoleARN": "arn:aws:iam::123456789012:role/
ScheduledQueryRole",
            "actionLogLineRoleArn": "arn:aws:iam::123456789012:role/LogLinesRole",
            "actionLogLineCount": 5,
            "schedule": {
                "expression": "rate(5 minutes)",
                "startTimeOffset": 300,
                "endTimeOffset": 0
            },
            "threshold": 100.0,
            "comparisonOperator": "GreaterThanThreshold",
            "treatMissingData": "missing",
            "queryResultsToEvaluate": 5,
            "queryResultsToAlarm": 3,
            "alarmName": "HighErrorRate",
            "description": "Alarm when error count exceeds 100",
            "actionsEnabled": true,
            "timestamp": "2026-06-10T12:00:00.000+0000",
            "okActions": [],
            "alarmActions": ["arn:aws:sns:us-east-1:123456789012:my-topic"],
            "insufficientDataActions": []
        }
    }
}
```

Alarm tags in action payloads

With CloudWatch, you can include an alarm's tags in the payloads it sends when the alarm changes state. You can then route, filter, or add context to notifications by using tags such as team, environment, or cost center.

You can use alarm tags with the following destinations: Amazon SNS notifications, Lambda function invocations, and the alarm state change events that CloudWatch sends to EventBridge.

CloudWatch sends alarm state change events to EventBridge by default, so configured tags appear in those events even when you don't configure an action on the alarm. For Amazon SNS and Lambda, tags appear only when you configure that action for the alarm. For Amazon SNS, CloudWatch includes the tags in the notification message that it delivers to the topic's subscribers, including those that use the HTTP/HTTPS and email protocols.

> **Note**
>
> CloudWatch doesn't support alarm tags for other actions, such as Systems Manager OpsItems, Systems Manager Incident Manager incidents, Amazon EC2 actions, Amazon EC2 Auto Scaling and Application Auto Scaling actions, or CloudWatch investigations.

Tag keys that begin with `aws:` are reserved for AWS and are never propagated.

Each payload includes a status field (`AlarmTagsStatus` for Amazon SNS, and `alarmTagsStatus` for Lambda and EventBridge) with one of the following values:

- `SUCCESS` – The tags were retrieved and any key filter was applied. The list can be empty if the filter excluded all tags.
- `NO_TAGS` – The alarm has no tags.
- `TRUNCATED` – Some tags were omitted to keep the payload within its size limit. If your tags are truncated, narrow the set of tags that CloudWatch includes by using the include or exclude tag key options.
- `FAILURE` – The tags couldn't be retrieved when the notification was sent. CloudWatch still delivers the notification or event, with no tags.

The representation of the tags differs by destination. Amazon SNS and Lambda include the tags as a list of key-value objects, and EventBridge includes them as key-value pairs to make event pattern filtering easier. For the Amazon SNS schema, see Schema of Amazon SNS notifications when alarms change state. For Lambda, see Invoke a Lambda function from an alarm. For EventBridge, see Alarm events and EventBridge.

Each notification or event reflects the alarm's tags at the time of the state change, including tag changes that you make after you create the alarm.

By default, when tag propagation is enabled, CloudWatch includes all of the alarm's tags (future tags as well if they are added later). To include only the relevant tags, or to exclude sensitive tags that you don't want to send in the payload, provide an include or exclude tag key list:

- Exclude tag keys – CloudWatch includes all of the alarm's tags except the keys that you specify. Any tags that you add later are also included, unless their keys are in the exclude list.

- Include tag keys – CloudWatch includes only the keys that you specify. If you add a new tag later and want it in the payload, you must add its key to the include list.

Prerequisites

A principal that creates or updates an alarm to enable this configuration must have the `cloudwatch:ListTagsForResource` permission for that alarm. CloudWatch also provides the `cloudwatch:AlarmTagsPropagationEnabled` condition key to control whether principals can enable tag propagation.

Enable alarm tag propagation using the console

You enable tag propagation while you create or edit an alarm in the CloudWatch console.

To enable alarm tag propagation

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then start creating an alarm, or edit an existing alarm.
2. On the **Add alarm details** step, under the tag editor, select **Include this alarm's tags in action and event payloads**.
3. To set which tags to include, select **All tags** to include all of the alarm's tags, or select **Select tags** to specify tag keys.
4. If you selected **Select tags**, select **Include tags** to include only the keys that you specify, or select **Exclude tags** to include all tags except the keys that you specify.
5. If you selected **Include tags** or **Exclude tags**, enter the tag keys in **Tag keys to include** or **Tag keys to exclude**.
6. Choose **Preview and create** to review your choice.
7. Finish creating or saving the alarm.

After you complete these steps, the alarm includes its tags in its notifications and events when its state changes.

To view the setting later, open the alarm's **Tags** tab on the alarm details page. To change the setting without the wizard, choose **Manage tags**.

Enable alarm tag propagation using the API

To enable tag propagation with the API, set the `AlarmTagsPropagation` configuration within the `ActionConfiguration` parameter on `PutMetricAlarm`, `PutCompositeAlarm`, or `PutLogAlarm`, and set `Enabled` to `true`. `DescribeAlarms` returns the current configuration.

To include all of the alarm's tags, omit both of the following mutually exclusive lists. To narrow the set of tags, use one of them:

- `IncludedTagKeys` – Include only the listed keys.
- `ExcludedTagKeys` – Include all keys except those listed.

Each list can contain from 1 to 50 tag keys, and each tag key can be from 1 to 128 characters.

For more information, see PutMetricAlarm, PutCompositeAlarm, PutLogAlarm, and DescribeAlarms in the *Amazon CloudWatch API Reference*.

Configure alarm mute rules

The steps in this section explain how to use the CloudWatch console to create an alarm mute rule. You can also use the API or AWS CLI to create an alarm mute rule. For more information, see PutAlarmMuteRule.

To create an alarm mute rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **Mute Rules** tab
3. Click **Create Alarm Mute Rule** button, which would open a wizard
4. Under **Alarm mute rule details** section, enter a name and optional description for your alarm mute rule
5. Under **Schedule pattern** section, choose one time or recurring schedule
    a. If you choose one-time occurrence,
        i. Select the Timezone to apply the mute schedules
        ii. Choose a start date and time, to define when the mute rule should become active
        iii. Choose end date and time, to define when the mute rule should expire

b. If you choose Recurring schedule, you will have two options. Either to use the Console form or use cron expression to configure the recurring time schedules.

   i. Under **Schedule creation type** choose "Specify date, time and recurrence" to use the Console form

      A. Choose the Timezone to apply the mute rule

      B. Choose **Start date and time**, to define when the mute rule should become active

      C. Choose **Duration**, to define how long the mute rule should last once becomes active

      D. Choose **Repeat**, to define how the schedule should repeat like every day, every month, every weekend or on specific days during the week.

      E. Choose optional **Until**, to define when the mute schedule should expire. Default option is "Indefinitely"

   ii. Under **Schedule creation type** choose "Set from a cron expression" to configure schedules using cron expressions

      A. Under **Cron expression** section enter the desired cron expression values.

      B. Choose **Duration**, to define how long the mute rule should last once becomes active

      C. Under optional **Timeframe** section, enter optional start and end date and time to define when the mute schedule should become active and expire.

6. Under **Target alarms** section, choose the alarms from the drop down to which you want to apply this mute rule

7. Under **Set tags for your mute rule** section, attach tags to your alarm mute rule. A tag is a key-value pair applied to a resource to hold metadata about that resource. For more information see What are tags?

8. Select **Create alarm mute rule** button to create the mute rule

Quick mutes

Alarms could be added for a short time period as follows,

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Alarms**

3. Select the alarms that you want to mute from the list
4. Under **Actions** menu choose **Mute**
5. Under the **Quick mute** section choose the predefined time periods 15min, 1h, 3h or select **Mute until** to set desired time period
6. Click **Confirm** to mute the alarms immediately until the chosen time period

Add alarms to existing mute rules

Alarms could be added to existing mute rules as follows,

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**
3. Select the alarms that you want to mute from the list
4. Under **Actions** menu choose **Mute**
5. Choose **Apply existing mute rules**, which should open a wizard
6. Select the mute rules from the drop down to which you want to add the alarms
7. Click **Apply**

> **Note**
>
> Quick mute and adding an alarm to existing mute rules options are also available from the alarm details page. **Mute rules** tab in the details page displays all mute rules associated with the alarm.

Managing alarms

Topics

- Edit or delete a CloudWatch alarm
- Hide Auto Scaling alarms
- Alarms and tagging
- Viewing and managing muted alarms

Edit or delete a CloudWatch alarm

You can edit or delete an existing alarm.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action**, **Copy**.

To edit an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose the name of the alarm.
4. To add or remove tags, choose the **Tags** tab and then choose **Manage tags**.
5. To edit other parts of the alarm, choose **Actions**, **Edit**.

   The **Specify metric and conditions** page appears, showing a graph and other information about the metric and statistic that you selected.
6. To change the metric, choose **Edit**, choose the **All metrics** tab, and do one of the following:
   - Choose the service namespace that contains the metric that you want. Continue choosing options as they appear to narrow the choices. When a list of metrics appears, select the check box next to the metric that you want.
   - In the search box, enter the name of a metric, dimension, or resource ID and press Enter. Then choose one of the results and continue until a list of metrics appears. Select the check box next to the metric that you want.

   Choose **Select metric**.
7. To change other aspects of the alarm, choose the appropriate options. To change how many data points must be breaching for the alarm to go into `ALARM` state or to change how missing data is treated, choose **Additional configuration**.
8. Choose **Next**.
9. Under **Notification**, **Auto Scaling action**, and **EC2 action**, optionally edit the actions taken when the alarm is triggered. Then choose **Next**.
10. Optionally change the alarm description.

You can't change the name of an existing alarm. You can copy an alarm and give the new alarm a different name. To copy an alarm, select the check box next to the alarm name in the alarm list and choose **Action**, **Copy**.

11. Choose **Next**.
12. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Update alarm**.

To update an email notification list that was created using the Amazon SNS console

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. In the navigation pane, choose **Topics** and then select the ARN for your notification list (topic).
3. Do one of the following:
   - To add an email address, choose **Create subscription**. For **Protocol**, choose **Email**. For **Endpoint**, enter the email address of the new recipient. Choose **Create subscription**.
   - To remove an email address, choose the **Subscription ID**. Choose **Other subscription actions**, **Delete subscriptions**.
4. Choose **Publish to topic**.

To delete an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**.
3. Select the check box to the left of the name of the alarm, and choose **Actions**, **Delete**.
4. Choose **Delete**.

Hide Auto Scaling alarms

When you view your alarms in the AWS Management Console, you can hide the alarms related to both Amazon EC2 Auto Scaling and Application Auto Scaling. This feature is available only in the AWS Management Console.

To temporarily hide Auto Scaling alarms

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Alarms**, **All alarms**, and select **Hide Auto Scaling alarms**.

Alarms and tagging

Tags are key-value pairs that can help you organize and categorize your resources. You can also use them to scope user permissions by granting a user permission to access or change only resources with certain tag values. For more general information about tagging resources, see Tagging your AWS resources

The following list explains some details about how tagging works with CloudWatch alarms.

- To be able to set or update tags for a CloudWatch resource, you must be signed on to an account that has the `cloudwatch:TagResource` permission. For example, to create an alarm and set tags for it, you must have the `cloudwatch:TagResource` permission in addition to the `cloudwatch:PutMetricAlarm` (or `cloudwatch:PutLogAlarm` for log alarms) permission. You recommend that you make sure anyone in your organization who will create or update CloudWatch resources has the `cloudwatch:TagResource` permission.
- Tags can be used for tag-based authorization control. For example, IAM user or role permissions can include conditions to limit CloudWatch calls to specific resources based on their tags. However, keep in mind the following
  - Tags with names that start with `aws:` can't be used for tag-based authorization control.
  - Composite alarms do not support tag-based authorization control.

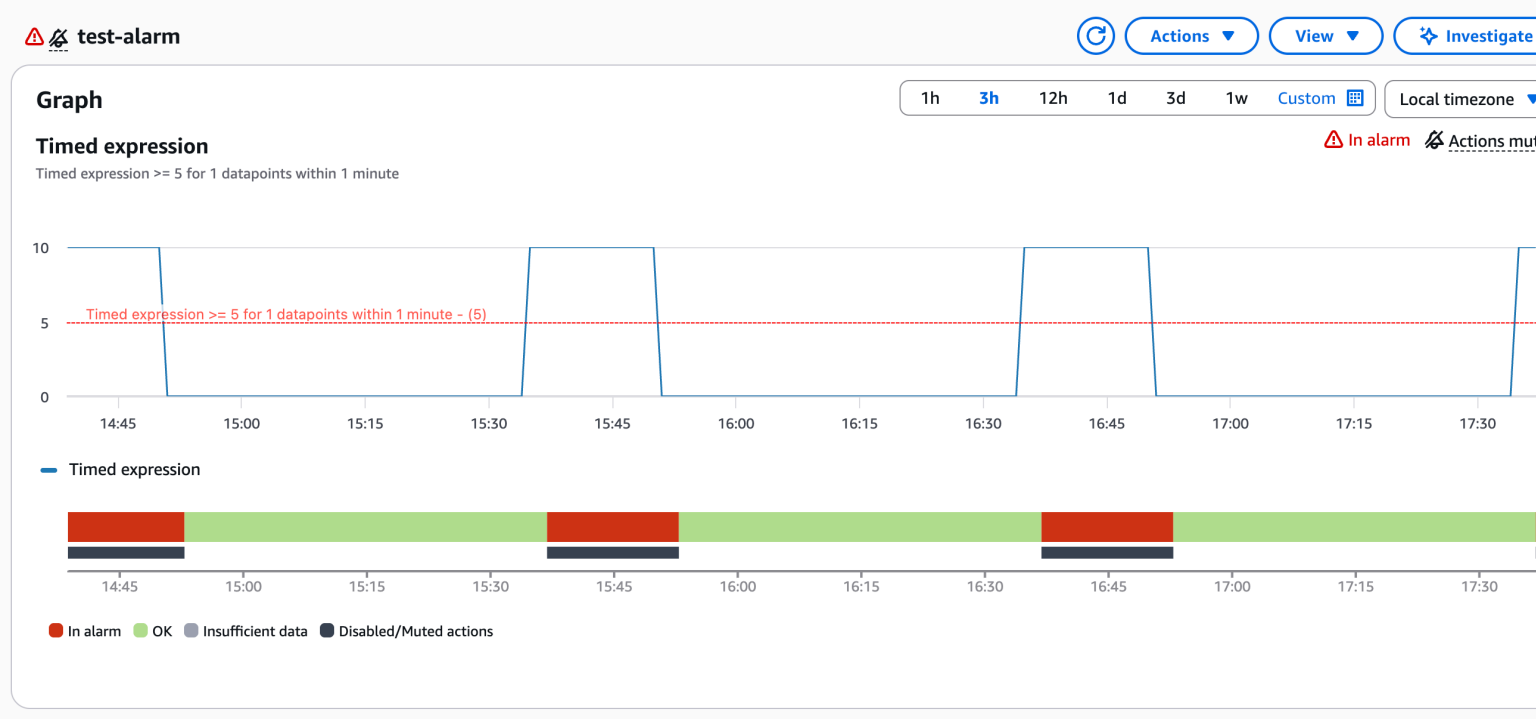
Viewing and managing muted alarms

Viewing muted alarms: You can easily identify which alarms are currently muted through the CloudWatch console. In both the alarms list view and individual alarm detail pages, a mute icon appears next to alarms whose actions are currently being muted by active mute rules. This visual indicator helps you quickly understand which alarms actions are being currently muted until the mute window expires.



Alarms timeline: The CloudWatch alarms console provides a comprehensive timeline view that shows when your alarm actions were muted. This timeline displays mute periods alongside alarm

state changes, giving you a complete historical view of both alarm behaviour and muting activity. You can use this timeline to analyze the effectiveness of your mute rules and understand how they correlate with your operational activities.



Programmatically checking alarm mute status: To programmatically determine if an alarm is currently muted, you can use the ListAlarmMuteRules API with the alarm name as a filter criteria. This API returns all active mute rules that are affecting the specified alarm, allowing you to integrate mute status checks into your automation workflows, monitoring dashboards, or operational tools.

For example: To check if an alarm named "HighCPUAlarm" is currently muted, call the ListAlarmMuteRules API with the filter parameter set to the alarm name. The response will include all mute rules targeting that alarm, along with their current status (SCHEDULED, ACTIVE, or EXPIRED).

Alarm history: Whenever alarm actions are muted due to an active mute rule, CloudWatch writes a history entry to the alarm's history log. This provides a complete audit trail of when your alarms were muted, helping you understand the timeline of muting events and correlate them with operational activities. You can view this history through the CloudWatch console or retrieve it programmatically using the DescribeAlarmHistory API.

Note

- When multiple alarm mute rules are active simultaneously, the most recently created mute rule name is written to the alarm history along with the total number of other active mute rules.
- The timeline displays mute periods only when an alarm state transitions during an active mute window and actions were prevented from executing.

Tip

You can manage alarm mute rules programmatically using the CloudWatch API. For more information, see PutAlarmMuteRule, GetAlarmMuteRule, ListAlarmMuteRules, and DeleteAlarmMuteRule.

Common features of CloudWatch alarms

The following features apply to all CloudWatch alarms:

- There is no limit to the number of alarms that you can create. To create or update an alarm, you use the CloudWatch console, the PutMetricAlarm or PutLogAlarm API actions, or the corresponding commands in the AWS CLI.
- Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters
- You can list any or all of the currently configured alarms, and list any alarms in a particular state by using the CloudWatch console, the DescribeAlarms API action, or the describe-alarms command in the AWS CLI.
- You can disable and enable alarm actions by using the DisableAlarmActions and EnableAlarmActions API actions, or the disable-alarm-actions and enable-alarm-actions commands in the AWS CLI.
- You can test an alarm by setting it to any state using the SetAlarmState API action or the set-alarm-state command in the AWS CLI. This temporary state change lasts only until the next alarm comparison occurs.
- You can create an alarm for a custom metric before you've created that custom metric. For the alarm to be valid, you must include all of the dimensions for the custom metric in addition

to the metric namespace and metric name in the alarm definition. To do this, you can use the PutMetricAlarm API action, or the put-metric-alarm command in the AWS CLI.

- You can view an alarm's history using the CloudWatch console, the DescribeAlarmHistory API action, or the describe-alarm-history command in the AWS CLI. CloudWatch preserves alarm history for 30 days. Each state transition is marked with a unique timestamp. In rare cases, your history might show more than one notification for a state change. The timestamp enables you to confirm unique state changes.
- You can favorite alarms from the *Favorites and recents* option in the CloudWatch console navigation pane by hovering over the alarm that you want to favorite and choosing the star symbol next to it.
- Alarms have an evaluation period quota. The evaluation period is calculated by multiplying the alarm period by the number of evaluation periods used.
  - The maximum evaluation period is seven days for alarms with a period of at least one hour (3600 seconds).
  - The maximum evaluation period is one day for alarms with a shorter period.
  - The maximum evaluation period is one day for alarms that use the custom Lambda data source.

> **Note**
>
> Some AWS resources don't send metric data to CloudWatch under certain conditions. For example, Amazon EBS might not send metric data for an available volume that is not attached to an Amazon EC2 instance, because there is no metric activity to be monitored for that volume. If you have an alarm set for such a metric, you might notice its state change to `INSUFFICIENT_DATA`. This might indicate that your resource is inactive, and might not necessarily mean that there is a problem. You can specify how each alarm treats missing data. For more information, see Configuring how CloudWatch alarms treat missing data.

Best practice alarm recommendations for AWS services

CloudWatch provides out-of-the box alarm recommendations. These are CloudWatch alarms that CloudWatch recommends that you create for metrics that are published by other AWS services. These recommendations can help you identify the metrics that you should set alarms for to follow

best practices for monitoring. The recommendations also suggest the alarm thresholds to set. Following these recommendations can help you not miss important monitoring of your AWS infrastructure.

To find the alarm recommendations, you use the metrics section of the CloudWatch console, and select the alarm recommendations filter toggle. If you navigate to the recommended alarms in the console and then create a recommended alarm, CloudWatch can pre-fill some of the alarm settings. For some recommended alarms, the alarm threshold value is also pre-filled. You can also use the console to download infrastructure-as-code alarm definitions for recommended alarms, and then use this code to create the alarm in AWS CloudFormation, the AWS CLI, or Terraform.

You can also see the list of recommended alarms in Recommended alarms (Classic).

You are charged for the alarms that you create, at the same rate as any other alarms that you create in CloudWatch. Using the recommendations incurs no extra charges. For more information, see Amazon CloudWatch Pricing.

Find and create recommended alarms

Follow these steps to find the metrics that CloudWatch recommends that you set alarms for, and optionally to create one of these alarms. The first procedure explains how to find the metrics that have recommended alarms, and how to create one of these alarms.

You can also get a bulk download of infrastructure-as-code alarm definitions for all recommended alarms in an AWS namespace, such as `AWS/Lambda` or `AWS/S3`. Those instructions are later in this topic.

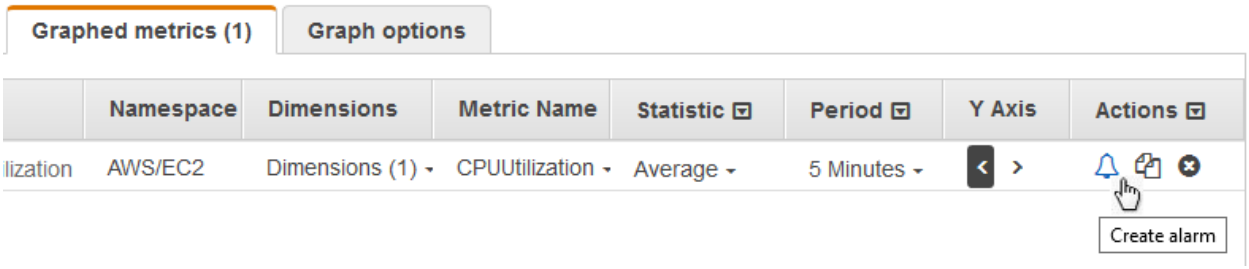
To find the metrics with recommended alarms, and create a single recommended alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Above the **Metrics** table, Choose **Alarm recommendations**.

   The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.
4. Choose the namespace for a service.

   The list of metrics under this namespace is filtered to include only those that have alarm recommendations.

5. To see the alarm intent and recommended threshold for a metric, choose **View details**.
6. To create an alarm for one of the metrics, do one of the following:
   - To use the console to create the alarm, do the following:
     a. Select the checkbox for the metric and choose the **Graphed metrics** tab.
     b. Choose the alarm icon.



     The alarm creation wizard appears, with the metric name, statistic, and period filled in based on the alarm recommendation. If the recommendation includes a specific threshold value, that value is also pre-filled.
     c. Choose **Next**.
     d. Under **Notification**, select an SNS topic to notify when the alarm transitions to ALARM state, OK state, or INSUFFICIENT_DATA state.

     To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

     To have the alarm not send notifications, choose **Remove**.
     e. To have the alarm perform Auto Scaling or EC2 actions, choose the appropriate button and choose the alarm state and action to perform.
     f. When finished, choose **Next**.
     g. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.
     h. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.
   - To download an infrastructure-as-code alarm definition to use in either AWS CloudFormation, AWS CLI, or Terraform, choose **Download alarm code** and select the format that you want. The downloaded code will have the recommended settings for the metric name, statistic, and threshold.

To download infrastructure-as-code alarm definitions for all recommended alarms for an AWS service

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, **All metrics**.
3. Above the **Metrics** table, Choose **Alarm recommendations**.

   The list of metric namespaces is filtered to include only the metrics that have alarm recommendations and that services in your account are publishing.
4. Choose the namespace for a service.

   The list of metrics under this namespace is filtered to include only those that have alarm recommendations.
5. The **Download alarm code** displays how many alarms are recommended for the metrics in this namespace. To download infrastructure-as-code alarm definitions for all recommended alarms, choose **Download alarm code** and then choose the code format that you want.

Recommended PromQL alarms

Use these PromQL alarms as equivalents of the classic recommended alarms. They monitor the same metrics by using PromQL instant queries that are evaluated against metrics ingested through the CloudWatch OTLP endpoint.

PromQL alarms use `EvaluationCriteria` with `PromQLCriteria`. Unlike classic metric alarms, the threshold is embedded directly in the PromQL query expression.

- **Pending period** – The duration in seconds that a time series must continuously breach before the alarm transitions to ALARM state.
- **Recovery period** – The duration in seconds that all time series must stop breaching before the alarm returns to OK state.
- **Evaluation interval** – How often, in seconds, CloudWatch runs the PromQL query.

> **Note**
>
> These alarms require metrics published through the CloudWatch OTLP endpoint. For more information, see PromQL alarms.

You can deploy these alarms by using AWS CloudFormation. Use the `EvaluationCriteria` and `PromQLCriteria` properties on the `AWS::CloudWatch::Alarm` resource.

Topics

- API Gateway
- Auto Scaling
- Certificate Manager
- CloudFront
- Cognito
- DynamoDB
- EBS
- Amazon Elastic Compute Cloud
- Amazon ECS
- Amazon ECS Container Insights
- Amazon ECS Container Insights enhanced observability
- Amazon EFS
- Amazon EKS Container Insights
- Amazon ElastiCache
- Elastic GPUs
- Amazon EventBridge Scheduler
- Amazon Kinesis Data Streams
- AWS Lambda
- AWS Lambda Insights
- NAT Gateway
- AWS PrivateLink endpoints
- AWS PrivateLink services
- RDS
- Route 53
- S3
- S3 Object Lambda
- SNS

- SQS
- VPN

API Gateway

REST API

4XXError

Labels: ApiName, Stage

Alarm description: Alarm when the average 4XX error rate exceeds 5% for a REST API stage.

Intent: Detect high client error rate indicating request issues.

PromQL criteria: `avg({__name__="4XXError", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiName="your-api-name",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% client error rate, which indicates significant request failures.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

5XXError

Labels: ApiName, Stage

Alarm description: Alarm when the average 5XX error rate exceeds 5% for a REST API stage.

Intent: Detect high server error rate indicating backend failures.

PromQL criteria: `avg({__name__="5XXError", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiName="your-api-name",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% server error rate, which requires immediate investigation.

Evaluation interval: 60

Pending period: 180

Recovery period: 300

Latency

Labels: ApiName, Stage

Alarm description: Alarm when the p90 latency exceeds 2500 ms for a REST API stage.

Intent: Detect high p90 latency affecting user experience.

PromQL criteria: `histogram_quantile(0.9, {__name__="Latency", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiName="your-api-name",Stage="your-stage"}) > 2500`

Recommended threshold: 2500 (embedded in query)

Threshold justification: A p90 latency above 2500 ms indicates degraded response times for most requests.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Count

Labels: ApiName, Stage

Alarm description: Alarm when the total request count drops below the expected baseline for a REST API stage.

Intent: Detect low traffic volume that might indicate routing or availability issues.

PromQL criteria: `sum({__name__="Count", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiName="your-api-name",Stage="your-stage"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your expected traffic baseline to detect unexpected drops.

Evaluation interval: 60

Pending period: 600

Recovery period: 300

HTTP API

4xx

Labels: ApiId, Stage

Alarm description: Alarm when the average 4xx error rate exceeds 5% for an HTTP API stage.

Intent: Detect high client error rate on HTTP API endpoints.

PromQL criteria: `avg({__name__="4xx", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% client error rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

5xx

Labels: ApiId, Stage

Alarm description: Alarm when the average 5xx error rate exceeds 5% for an HTTP API stage.

Intent: Detect high server error rate on HTTP API endpoints.

PromQL criteria: `avg({__name__="5xx", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% server error rate requiring investigation.

Evaluation interval: 60

Pending period: 180

Recovery period: 300

Latency

Labels: ApiId, Stage

Alarm description: Alarm when the p90 latency exceeds 2500 ms for an HTTP API stage.

Intent: Detect high p90 latency on HTTP API endpoints.

PromQL criteria: `histogram_quantile(0.9, {__name__="Latency", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 2500`

Recommended threshold: 2500 (embedded in query)

Threshold justification: A p90 latency above 2500 ms indicates degraded response times.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

IntegrationLatency

Labels: ApiId, Stage

Alarm description: Alarm when the p90 integration latency exceeds 2000 ms for an HTTP API stage.

Intent: Detect high backend integration latency affecting response times.

PromQL criteria: `histogram_quantile(0.9, {__name__="IntegrationLatency", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 2000`

Recommended threshold: 2000 (embedded in query)

Threshold justification: A p90 integration latency above 2000 ms indicates backend slowness.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Count

Labels: ApiId, Stage

Alarm description: Alarm when the total request count drops below the expected baseline for an HTTP API stage.

Intent: Detect low traffic volume that might indicate routing or availability issues.

PromQL criteria: `sum({__name__="Count", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your expected traffic baseline to detect unexpected drops.

Evaluation interval: 60

Pending period: 600

Recovery period: 300

WebSocket API

ClientError

Labels: ApiId, Stage

Alarm description: Alarm when the average client error rate exceeds 5% for a WebSocket API stage.

Intent: Detect high client error rate on WebSocket API connections.

PromQL criteria: `avg({__name__="ClientError", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% client error rate on WebSocket connections.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ExecutionError

Labels: ApiId, Stage

Alarm description: Alarm when the average execution error rate exceeds 5% for a WebSocket API stage.

Intent: Detect high server-side execution error rate on WebSocket APIs.

PromQL criteria: `avg({__name__="ExecutionError", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Detects greater than 5% execution error rate requiring investigation.

Evaluation interval: 60

Pending period: 180

Recovery period: 300

IntegrationLatency

Labels: ApiId, Stage

Alarm description: Alarm when the p90 integration latency exceeds 2000 ms for a WebSocket API stage.

Intent: Detect high backend integration latency on WebSocket API routes.

PromQL criteria: `histogram_quantile(0.9, {__name__="IntegrationLatency", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) > 2000`

Recommended threshold: 2000 (embedded in query)

Threshold justification: A p90 integration latency above 2000 ms indicates backend slowness.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

MessageCount

Labels: ApiId, Stage

Alarm description: Alarm when the total message count drops below the expected baseline for a WebSocket API stage.

Intent: Detect low message volume that might indicate connection or routing issues.

PromQL criteria: `sum({__name__="MessageCount", "@instrumentation.@name"="cloudwatch.aws/apigateway", ApiId="your-api-id",Stage="your-stage"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your expected message volume to detect unexpected drops.

Evaluation interval: 60

Pending period: 600

Recovery period: 300

Auto Scaling

GroupInServiceCapacity

Labels: AutoScalingGroupName

Alarm description: Alarm when the average in-service capacity drops below the expected minimum for an Auto Scaling group.

Intent: Detect low availability due to launch failures or terminated instances.

PromQL criteria: `avg({__name__="GroupInServiceCapacity", "@instrumentation.@name"="cloudwatch.aws/autoscaling", AutoScalingGroupName="`*`your-auto-scaling-group-name`*`"}) < `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your minimum desired capacity to detect launch failures or insufficient instances.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

Certificate Manager

DaysToExpiry

Labels: CertificateArn

Alarm description: Alarm when the minimum days to certificate expiry drops to 44 days or fewer.

Intent: Proactive certificate expiration alert to allow time for renewal.

PromQL criteria: `min({__name__="DaysToExpiry", "@instrumentation.@name"="cloudwatch.aws/certificatemanager", CertificateArn="`*`your-certificate-arn`*`"}) <= 44`

Recommended threshold: 44 (embedded in query)

Threshold justification: Provides sufficient lead time before certificate expiration to complete the renewal process.

Evaluation interval: 86400

Pending period: 86400

Recovery period: 86400

CloudFront

5xxErrorRate

Labels: DistributionId

Alarm description: Alarm when the average 5xx error rate exceeds the threshold for a CloudFront distribution.

Intent: Detect high origin or CloudFront error rate affecting content delivery.

PromQL criteria: `avg({__name__="5xxErrorRate", "@instrumentation.@name"="cloudwatch.aws/cloudfront", DistributionId="`*`your-distribution-id`*`"}) > `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your acceptable error rate for content delivery.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

OriginLatency

Labels: DistributionId

Alarm description: Alarm when the p90 origin latency exceeds the threshold for a CloudFront distribution.

Intent: Detect high origin response latency affecting content delivery performance.

PromQL criteria: `histogram_quantile(0.9, {__name__="OriginLatency", "@instrumentation.@name"="cloudwatch.aws/cloudfront", DistributionId="your-distribution-id"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your expected origin response time and caching strategy.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

FunctionValidationErrors

Labels: DistributionId, FunctionName

Alarm description: Alarm when any CloudFront function validation errors occur.

Intent: Detect CloudFront function validation failures that prevent function execution.

PromQL criteria: `sum({__name__="FunctionValidationErrors", "@instrumentation.@name"="cloudwatch.aws/cloudfront", DistributionId="your-distribution-id",FunctionName="your-function-name"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any validation error indicates a function configuration problem that requires attention.

Evaluation interval: 60

Pending period: 120

Recovery period: 120

FunctionExecutionErrors

Labels: DistributionId, FunctionName

Alarm description: Alarm when any CloudFront function execution errors occur.

Intent: Detect runtime failures in CloudFront functions that affect request processing.

PromQL criteria: `sum({__name__="FunctionExecutionErrors", "@instrumentation.@name"="cloudwatch.aws/cloudfront", DistributionId="your-distribution-id",FunctionName="your-function-name"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any execution error indicates a runtime problem in the function code.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

FunctionThrottles

Labels: DistributionId, FunctionName

Alarm description: Alarm when any CloudFront function throttling occurs.

Intent: Detect CloudFront function throttling that might degrade request processing.

PromQL criteria: `sum({__name__="FunctionThrottles", "@instrumentation.@name"="cloudwatch.aws/cloudfront", DistributionId="your-distribution-id",FunctionName="your-function-name"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any throttling indicates the function is hitting compute limits.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Cognito

SignUpThrottles

Labels: UserPool, UserPoolClient

Alarm description: Alarm when sign-up throttle events exceed the threshold for a user pool.

Intent: Detect sign-up throttling that prevents new user registrations.

PromQL criteria: `sum({__name__="SignUpThrottles", "@instrumentation.@name"="cloudwatch.aws/cognito", UserPool="your-user-pool",UserPoolClient="your-user-pool-client"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle rate for sign-up operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

SignInThrottles

Labels: UserPool, UserPoolClient

Alarm description: Alarm when sign-in throttle events exceed the threshold for a user pool.

Intent: Detect sign-in throttling that prevents user authentication.

PromQL criteria: `sum({__name__="SignInThrottles", "@instrumentation.@name"="cloudwatch.aws/cognito", UserPool="your-user-pool",UserPoolClient="your-user-pool-client"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle rate for sign-in operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TokenRefreshThrottles

Labels: UserPool, UserPoolClient

Alarm description: Alarm when token refresh throttle events exceed the threshold for a user pool.

Intent: Detect token refresh throttling that might cause session interruptions.

PromQL criteria: `sum({__name__="TokenRefreshThrottles", "@instrumentation.@name"="cloudwatch.aws/cognito", UserPool="your-user-pool",UserPoolClient="your-user-pool-client"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle rate for token refresh operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

FederationThrottles

Labels: UserPool, UserPoolClient, IdentityProvider

Alarm description: Alarm when federation throttle events exceed the threshold for a user pool identity provider.

Intent: Detect federation throttling that might prevent federated sign-in.

PromQL criteria: `sum({__name__="FederationThrottles", "@instrumentation.@name"="cloudwatch.aws/cognito", UserPool="your-user-pool",UserPoolClient="your-user-pool-client",IdentityProvider="your-identity-provider"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle rate for federation operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

DynamoDB

AccountProvisionedReadCapacityUtilization

Labels: None

Alarm description: Alarm when the account-level provisioned read capacity utilization exceeds 80%.

Intent: Detect when provisioned read capacity approaches account limits.

PromQL criteria: `max({__name__="AccountProvisionedReadCapacityUtilization", "@instrumentation.@name"="cloudwatch.aws/dynamodb"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: At 80% utilization, you have limited headroom before reaching account limits.

Evaluation interval: 300

Pending period: 600

Recovery period: 600

AccountProvisionedWriteCapacityUtilization

Labels: None

Alarm description: Alarm when the account-level provisioned write capacity utilization exceeds 80%.

Intent: Detect when provisioned write capacity approaches account limits.

PromQL criteria: `max({__name__="AccountProvisionedWriteCapacityUtilization", "@instrumentation.@name"="cloudwatch.aws/dynamodb"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: At 80% utilization, you have limited headroom before reaching account limits.

Evaluation interval: 300

Pending period: 600

Recovery period: 600

AgeOfOldestUnreplicatedRecord

Labels: TableName, DelegatedOperation

Alarm description: Alarm when the age of the oldest unreplicated record exceeds the threshold.

Intent: Detect replication lag that might cause stale data in global tables.

PromQL criteria: `max({__name__="AgeOfOldestUnreplicatedRecord", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",DelegatedOperation="your-operation"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your replication freshness requirements.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

FailedToReplicateRecordCount

Labels: TableName, DelegatedOperation

Alarm description: Alarm when any records fail to replicate in a global table.

Intent: Detect replication failures that cause data inconsistency across regions.

PromQL criteria: `sum({__name__="FailedToReplicateRecordCount", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",DelegatedOperation="your-operation"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any failed replication indicates data consistency issues requiring immediate attention.

Evaluation interval: 60

Pending period: 60

Recovery period: 60

ReadThrottleEvents

Labels: TableName

Alarm description: Alarm when read throttle events exceed the threshold for a table.

Intent: Detect read throttling that indicates insufficient provisioned capacity.

PromQL criteria: `sum({__name__="ReadThrottleEvents", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle count for read operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ReadThrottleEvents (GSI)

Labels: TableName, GlobalSecondaryIndexName

Alarm description: Alarm when read throttle events exceed the threshold for a global secondary index.

Intent: Detect read throttling on a GSI that indicates insufficient index capacity.

PromQL criteria: `sum({__name__="ReadThrottleEvents", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",GlobalSecondaryIndexName="your-index-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle count for GSI read operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ReplicationLatency

Labels: TableName, ReceivingRegion

Alarm description: Alarm when the average replication latency exceeds the threshold for a global table.

Intent: Detect high replication latency that might cause stale reads in replica regions.

PromQL criteria: `avg({__name__="ReplicationLatency", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",ReceivingRegion="your-receiving-region"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your data freshness requirements for replica regions.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

SuccessfulRequestLatency

Labels: TableName, Operation

Alarm description: Alarm when the average successful request latency exceeds the threshold for a table operation.

Intent: Detect high latency on DynamoDB operations affecting application performance.

PromQL criteria: `avg({__name__="SuccessfulRequestLatency", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",Operation="your-operation"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your latency SLA for the specific DynamoDB operation.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

SystemErrors

Labels: TableName

Alarm description: Alarm when system errors exceed the threshold for a table.

Intent: Detect DynamoDB service-side errors that affect table availability.

PromQL criteria: `sum({__name__="SystemErrors", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable system error count for the table.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

ThrottledPutRecordCount

Labels: TableName, DelegatedOperation

Alarm description: Alarm when throttled put record count exceeds the threshold for a table.

Intent: Detect throttled puts that might cause data loss in streaming operations.

PromQL criteria: `max({__name__="ThrottledPutRecordCount", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",DelegatedOperation="your-operation"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle count for streaming put operations.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

UserErrors

Labels: None

Alarm description: Alarm when user errors exceed the threshold across the account.

Intent: Detect high rates of client errors such as validation or conditional check failures.

PromQL criteria: `sum({__name__="UserErrors", "@instrumentation.@name"="cloudwatch.aws/dynamodb"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable error rate for client-side request failures.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

WriteThrottleEvents

Labels: TableName

Alarm description: Alarm when write throttle events exceed the threshold for a table.

Intent: Detect write throttling that indicates insufficient provisioned capacity.

PromQL criteria: `sum({__name__="WriteThrottleEvents", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle count for write operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

WriteThrottleEvents (GSI)

Labels: TableName, GlobalSecondaryIndexName

Alarm description: Alarm when write throttle events exceed the threshold for a global secondary index.

Intent: Detect write throttling on a GSI that indicates insufficient index capacity.

PromQL criteria: `sum({__name__="WriteThrottleEvents", "@instrumentation.@name"="cloudwatch.aws/dynamodb", TableName="your-table-name",GlobalSecondaryIndexName="your-index-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your acceptable throttle count for GSI write operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

EBS

VolumeStalledIOCheck

Labels: VolumeId, InstanceId

Alarm description: Alarm when an EBS volume reports a stalled I/O check failure.

Intent: Detect stalled I/O on EBS volumes that indicates volume impairment.

PromQL criteria: `max({__name__="VolumeStalledIOCheck", "@instrumentation.@name"="cloudwatch.aws/ebs", VolumeId="your-volume-id",InstanceId="your-instance-id"}) >= 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: A value of 1 or greater indicates the volume has stalled I/O, requiring immediate investigation.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

Amazon Elastic Compute Cloud

CPUUtilization

Labels: InstanceId

Alarm description: Alarm when average CPU utilization exceeds 80% for an EC2 instance.

Intent: Detect high CPU utilization.

PromQL criteria: `avg({__name__="CPUUtilization", "@instrumentation.@name"="cloudwatch.aws/ec2", InstanceId="your-instance-id"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% threshold gives headroom before saturation.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

StatusCheckFailed

Labels: InstanceId

Alarm description: Alarm when an instance or system health check fails for an EC2 instance.

Intent: Detect instance or system health issues.

PromQL criteria: `max({__name__="StatusCheckFailed", "@instrumentation.@name"="cloudwatch.aws/ec2", InstanceId="your-instance-id"}) >= 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: Any status check failure requires investigation.

Evaluation interval: 300

Pending period: 600

Recovery period: 600

StatusCheckFailed_AttachedEBS

Labels: InstanceId

Alarm description: Alarm when an attached EBS volume becomes unreachable for an EC2 instance.

Intent: Detect unreachable EBS volumes.

PromQL criteria: `max({__name__="StatusCheckFailed_AttachedEBS", "@instrumentation.@name"="cloudwatch.aws/ec2", InstanceId="your-instance-id"}) >= 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: Unreachable volumes cause I/O failures.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

Amazon ECS

CPUReservation

Labels: ClusterName

Alarm description: Alarm when average CPU reservation exceeds 80% for an ECS cluster.

Intent: Detect cluster approaching CPU capacity.

PromQL criteria: `avg({__name__="CPUReservation", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% reservation indicates limited headroom for new tasks.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

CPUUtilization

Labels: ClusterName, ServiceName

Alarm description: Alarm when average CPU utilization exceeds 80% for an ECS service.

Intent: Detect high CPU utilization for ECS service.

PromQL criteria: `avg({__name__="CPUUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

EBSFilesystemUtilization

Labels: ClusterName, ServiceName

Alarm description: Alarm when average EBS filesystem utilization exceeds 80% for an ECS service.

Intent: Detect high EBS storage utilization.

PromQL criteria: `avg({__name__="EBSFilesystemUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

MemoryReservation

Labels: ClusterName

Alarm description: Alarm when average memory reservation exceeds 80% for an ECS cluster.

Intent: Detect cluster approaching memory capacity.

PromQL criteria: `avg({__name__="MemoryReservation", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% reservation indicates limited headroom for new tasks.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

MemoryUtilization

Labels: ClusterName, ServiceName

Alarm description: Alarm when average memory utilization exceeds 80% for an ECS service.

Intent: Detect high memory utilization.

PromQL criteria: `avg({__name__="MemoryUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

HTTPCode_Target_5XX_Count

Labels: ClusterName, ServiceName

Alarm description: Alarm when the HTTP 5XX error count exceeds the threshold for an ECS service.

Intent: Detect high server-side error count.

PromQL criteria: `sum({__name__="HTTPCode_Target_5XX_Count", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your application's normal error rate baseline.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TargetResponseTime

Labels: ClusterName, ServiceName

Alarm description: Alarm when average target response time exceeds the threshold for an ECS service.

Intent: Detect high target response time.

PromQL criteria: `avg({__name__="TargetResponseTime", "@instrumentation.@name"="cloudwatch.aws/ecs", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your application's acceptable latency requirements.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Amazon ECS Container Insights

EphemeralStorageUtilized

Labels: ClusterName, ServiceName

Alarm description: Alarm when average ephemeral storage usage exceeds the threshold for Fargate tasks.

Intent: Detect high ephemeral storage usage for Fargate tasks.

PromQL criteria: `avg({__name__="EphemeralStorageUtilized", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your task's ephemeral storage allocation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

RunningTaskCount

Labels: ClusterName, ServiceName

Alarm description: Alarm when the running task count drops to zero for an ECS service.

Intent: Detect when no tasks are running.

PromQL criteria: `avg({__name__="RunningTaskCount", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) <= 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Zero running tasks indicates a service outage.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

instance_filesystem_utilization

Labels: InstanceId, ContainerInstanceId, ClusterName

Alarm description: Alarm when average filesystem utilization exceeds 90% on a container instance.

Intent: Detect high filesystem utilization.

PromQL criteria: `avg({__name__="instance_filesystem_utilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", InstanceId="your-instance-id",ContainerInstanceId="your-container-instance-id",ClusterName="your-cluster-name"}) > 90`

Recommended threshold: 90 (embedded in query)

Threshold justification: 90% filesystem utilization leaves minimal space for operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Amazon ECS Container Insights enhanced observability

TaskCpuUtilization (cluster level)

Labels: ClusterName

Alarm description: Alarm when average task CPU utilization exceeds 80% at the cluster level.

Intent: Detect high CPU utilization.

PromQL criteria: `avg({__name__="TaskCpuUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskCpuUtilization (service level)

Labels: ClusterName, ServiceName

Alarm description: Alarm when average task CPU utilization exceeds 80% at the service level.

Intent: Detect high CPU utilization.

PromQL criteria: `avg({__name__="TaskCpuUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskMemoryUtilization (cluster level)

Labels: ClusterName

Alarm description: Alarm when average task memory utilization exceeds 80% at the cluster level.

Intent: Detect high memory utilization.

PromQL criteria: `avg({__name__="TaskMemoryUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskMemoryUtilization (service level)

Labels: ClusterName, ServiceName

Alarm description: Alarm when average task memory utilization exceeds 80% at the service level.

Intent: Detect high memory utilization.

PromQL criteria: `avg({__name__="TaskMemoryUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ContainerCpuUtilization (cluster level)

Labels: ClusterName

Alarm description: Alarm when average container CPU utilization exceeds 80% at the cluster level.

Intent: Detect high CPU utilization.

PromQL criteria: `avg({__name__="ContainerCpuUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ContainerCpuUtilization (container level)

Labels: ContainerName, ClusterName, ServiceName

Alarm description: Alarm when average container CPU utilization exceeds 80% at the container level.

Intent: Detect high CPU utilization.

PromQL criteria: `avg({__name__="ContainerCpuUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ContainerName="your-container-name",ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ContainerMemoryUtilization (cluster level)

Labels: ClusterName

Alarm description: Alarm when average container memory utilization exceeds 80% at the cluster level.

Intent: Detect high memory utilization.

PromQL criteria: `avg({__name__="ContainerMemoryUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ContainerMemoryUtilization (container level)

Labels: ContainerName, ClusterName, ServiceName

Alarm description: Alarm when average container memory utilization exceeds 80% at the container level.

Intent: Detect high memory utilization.

PromQL criteria: `avg({__name__="ContainerMemoryUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ContainerName="your-container-name",ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskEBSfilesystemUtilization

Labels: ClusterName, ServiceName

Alarm description: Alarm when average EBS filesystem utilization exceeds 80% for tasks.

Intent: Detect high EBS filesystem utilization.

PromQL criteria: `avg({__name__="TaskEBSfilesystemUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskEphemeralStorageUtilization (cluster level)

Labels: ClusterName

Alarm description: Alarm when average ephemeral storage utilization exceeds 80% at the cluster level.

Intent: Detect high ephemeral storage utilization.

PromQL criteria: `avg({__name__="TaskEphemeralStorageUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

TaskEphemeralStorageUtilization (service level)

Labels: ClusterName, ServiceName

Alarm description: Alarm when average ephemeral storage utilization exceeds 80% at the service level.

Intent: Detect high ephemeral storage utilization.

PromQL criteria: `avg({__name__="TaskEphemeralStorageUtilization", "@instrumentation.@name"="cloudwatch.aws/ecs-containerinsights", ClusterName="your-cluster-name",ServiceName="your-service-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Amazon EFS

PercentIOLimit

Labels: FileSystemId

Alarm description: Alarm when an EFS filesystem reaches 100% of its I/O limit.

Intent: Detect when filesystem reaches I/O limit.

PromQL criteria: `avg({__name__="PercentIOLimit", "@instrumentation.@name"="cloudwatch.aws/efs", FileSystemId="your-filesystem-id"}) >= 100`

Recommended threshold: 100 (embedded in query)

Threshold justification: At 100% the filesystem becomes a bottleneck.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

BurstCreditBalance

Labels: FileSystemId

Alarm description: Alarm when burst credit balance drops to zero for an EFS filesystem.

Intent: Detect depleted burst credits causing throughput slowdown.

PromQL criteria: `avg({__name__="BurstCreditBalance", "@instrumentation.@name"="cloudwatch.aws/efs", FileSystemId="your-filesystem-id"}) <= 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Zero credits causes reduced throughput.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

Amazon EKS Container Insights

node_cpu_utilization

Labels: ClusterName

Alarm description: Alarm when maximum CPU utilization exceeds 80% on EKS worker nodes.

Intent: Detect high CPU on worker nodes.

PromQL criteria: `max({__name__="node_cpu_utilization", "@instrumentation.@name"="cloudwatch.aws/eks-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

node_filesystem_utilization

Labels: ClusterName

Alarm description: Alarm when maximum filesystem utilization exceeds the threshold on EKS worker nodes.

Intent: Detect high filesystem usage on worker nodes.

PromQL criteria: `max({__name__="node_filesystem_utilization", "@instrumentation.@name"="cloudwatch.aws/eks-containerinsights", ClusterName="your-cluster-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on your node's storage capacity and usage patterns.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

node_memory_utilization

Labels: ClusterName

Alarm description: Alarm when maximum memory utilization exceeds 80% on EKS worker nodes.

Intent: Detect high memory on worker nodes.

PromQL criteria: `max({__name__="node_memory_utilization", "@instrumentation.@name"="cloudwatch.aws/eks-containerinsights", ClusterName="your-cluster-name"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

pod_cpu_utilization_over_pod_limit

Labels: ClusterName, Namespace, Service

Alarm description: Alarm when pod CPU utilization exceeds 80% of its limit.

Intent: Detect pods approaching CPU limits.

PromQL criteria: `max({__name__="pod_cpu_utilization_over_pod_limit", "@instrumentation.@name"="cloudwatch.aws/eks-containerinsights", ClusterName="your-cluster-name",Namespace="your-namespace",Service="your-service"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before throttling occurs.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

pod_memory_utilization_over_pod_limit

Labels: ClusterName, Namespace, Service

Alarm description: Alarm when pod memory utilization exceeds 80% of its limit.

Intent: Detect pods approaching memory limits.

PromQL criteria: `max({__name__="pod_memory_utilization_over_pod_limit", "@instrumentation.@name"="cloudwatch.aws/eks-containerinsights", ClusterName="your-cluster-name",Namespace="your-namespace",Service="your-service"}) > 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: 80% gives headroom before OOMKill occurs.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Amazon ElastiCache

CPUUtilization

Labels: CacheClusterId, CacheNodeId

Alarm description: Alarm when average CPU utilization exceeds the threshold for an ElastiCache node.

Intent: Detect high CPU across the instance.

PromQL criteria: `avg({__name__="CPUUtilization", "@instrumentation.@name"="cloudwatch.aws/elasticache", CacheClusterId="`*`your-cache-cluster-id`*`",CacheNodeId="`*`your-cache-node-id`*`"}) > `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your node type and workload characteristics.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

CurrConnections

Labels: CacheClusterId, CacheNodeId

Alarm description: Alarm when the connection count exceeds the threshold for an ElastiCache node.

Intent: Detect high connection counts.

PromQL criteria: `avg({__name__="CurrConnections", "@instrumentation.@name"="cloudwatch.aws/elasticache", CacheClusterId="`*`your-cache-cluster-id`*`",CacheNodeId="`*`your-cache-node-id`*`"}) > `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your expected connection pool size and application needs.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

DatabaseMemoryUsagePercentage

Labels: CacheClusterId

Alarm description: Alarm when database memory usage exceeds the threshold for an ElastiCache cluster.

Intent: Detect high memory usage to prevent write failures.

PromQL criteria: `avg({__name__="DatabaseMemoryUsagePercentage", "@instrumentation.@name"="cloudwatch.aws/elasticache", CacheClusterId="`*`your-cache-cluster-id`*`"}) > `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your eviction policy and data criticality.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

EngineCPUUtilization

Labels: CacheClusterId

Alarm description: Alarm when Redis engine CPU utilization exceeds 90% for an ElastiCache cluster.

Intent: Detect high Redis engine CPU utilization.

PromQL criteria: `avg({__name__="EngineCPUUtilization", "@instrumentation.@name"="cloudwatch.aws/elasticache", CacheClusterId="`*`your-cache-cluster-id`*`"}) > 90`

Recommended threshold: 90 (embedded in query)

Threshold justification: Redis is single-threaded; greater than 90% indicates saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ReplicationLag

Labels: CacheClusterId

Alarm description: Alarm when replication lag exceeds the threshold for an ElastiCache cluster.

Intent: Detect replication delay affecting data consistency.

PromQL criteria: `avg({__name__="ReplicationLag", "@instrumentation.@name"="cloudwatch.aws/elasticache", CacheClusterId="`*`your-cache-cluster-id`*`"}) > `*`THRESHOLD`*

Recommended threshold: *`THRESHOLD`* (embedded in query)

Threshold justification: Set based on your application's tolerance for stale reads.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

Elastic GPUs

GPUConnectivityCheckFailed

Labels: InstanceId, EGPUId

Alarm description: Alarm when the Elastic Graphics accelerator loses connectivity to the instance.

Intent: Detect connectivity issues to Elastic Graphics accelerator.

PromQL criteria: `max({__name__="GPUConnectivityCheckFailed", "@instrumentation.@name"="cloudwatch.aws/elasticgpus", InstanceId="your-instance-id",EGPUId="your-egpu-id"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any connectivity failure requires investigation.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

GPUHealthCheckFailed

Labels: InstanceId, EGPUId

Alarm description: Alarm when an Elastic Graphics accelerator health check fails.

Intent: Detect unhealthy Elastic Graphics accelerator.

PromQL criteria: `max({__name__="GPUHealthCheckFailed", "@instrumentation.@name"="cloudwatch.aws/elasticgpus", InstanceId="your-instance-id",EGPUId="your-egpu-id"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Health check failures indicate accelerator problems.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

Amazon EventBridge Scheduler

TargetErrorThrottledCount

Alarm description: Alarm when schedule target invocations are throttled.

Intent: Detect target throttling causing schedule delays.

PromQL criteria: `sum({__name__="TargetErrorThrottledCount", "@instrumentation.@name"="cloudwatch.aws/scheduler"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any throttling indicates target capacity issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

InvocationThrottleCount

Alarm description: Alarm when Scheduler invocations are throttled.

Intent: Detect Scheduler invocation throttling.

PromQL criteria: `sum({__name__="InvocationThrottleCount", "@instrumentation.@name"="cloudwatch.aws/scheduler"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Throttling delays scheduled executions.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

InvocationDroppedCount

Alarm description: Alarm when scheduled invocations are dropped.

Intent: Detect dropped invocations.

PromQL criteria: `sum({__name__="InvocationDroppedCount", "@instrumentation.@name"="cloudwatch.aws/scheduler"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Dropped invocations represent lost scheduled events.

Evaluation interval: 60

Pending period: 60

Recovery period: 60

InvocationsFailedToBeSentToDeadLetterCount

Alarm description: Alarm when failed invocations cannot be sent to the dead-letter queue.

Intent: Detect DLQ delivery failures.

PromQL criteria: `sum({__name__="InvocationsFailedToBeSentToDeadLetterCount", "@instrumentation.@name"="cloudwatch.aws/scheduler"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Failed DLQ delivery means lost error information.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

Labels: StreamName

Alarm description: Alarm when consumer iterator age exceeds the threshold for a Kinesis stream.

Intent: Detect data falling behind retention period risking data loss.

PromQL criteria: `max({__name__="GetRecords.IteratorAgeMilliseconds", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="`*`your-stream-name`*`"}) > `*`THRESHOLD`*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on stream retention period percentage.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

GetRecords.Success

Labels: StreamName

Alarm description: Alarm when the GetRecords success rate drops below the threshold for a Kinesis stream.

Intent: Detect consumer retrieval failures.

PromQL criteria: `avg({__name__="GetRecords.Success", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="your-stream-name"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on expected success rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

PutRecord.Success

Labels: StreamName

Alarm description: Alarm when the PutRecord success rate drops below the threshold for a Kinesis stream.

Intent: Detect producer ingestion failures.

PromQL criteria: `avg({__name__="PutRecord.Success", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="your-stream-name"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on expected success rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

PutRecords.FailedRecords

Labels: StreamName

Alarm description: Alarm when batch put operations produce failed records for a Kinesis stream.

Intent: Detect batch put failures.

PromQL criteria: `sum({__name__="PutRecords.FailedRecords", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="your-stream-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable failure count.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ReadProvisionedThroughputExceeded

Labels: StreamName

Alarm description: Alarm when consumer reads exceed provisioned throughput for a Kinesis stream.

Intent: Detect consumer read throttling.

PromQL criteria: `avg({__name__="ReadProvisionedThroughputExceeded", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="your-stream-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Any sustained throttling indicates capacity needs.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

SubscribeToShardEvent.MillisBehindLatest

Labels: StreamName, ConsumerName

Alarm description: Alarm when an enhanced fan-out consumer falls behind the latest record.

Intent: Detect enhanced fan-out consumer processing lag.

PromQL criteria: `avg({__name__="SubscribeToShardEvent.MillisBehindLatest", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="`*your-stream-name*`",ConsumerName="`*your-consumer-name*`"}) > `*THRESHOLD*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable processing delay.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

WriteProvisionedThroughputExceeded

Labels: StreamName

Alarm description: Alarm when producer writes exceed provisioned throughput for a Kinesis stream.

Intent: Detect producer write throttling.

PromQL criteria: `avg({__name__="WriteProvisionedThroughputExceeded", "@instrumentation.@name"="cloudwatch.aws/kinesis", StreamName="`*your-stream-name*`"}) > `*THRESHOLD*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Sustained throttling indicates capacity needs.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

AWS Lambda

ClaimedAccountConcurrency

Alarm description: Alarm when claimed account concurrency exceeds the threshold.

Intent: Detect account approaching concurrency quota.

PromQL criteria: `max({__name__="ClaimedAccountConcurrency", "@instrumentation.@name"="cloudwatch.aws/lambda"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set to percentage of regional concurrency limit.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

Errors

Labels: FunctionName

Alarm description: Alarm when function error count exceeds the threshold for a Lambda function.

Intent: Detect high function error count.

PromQL criteria: `sum({__name__="Errors", "@instrumentation.@name"="cloudwatch.aws/lambda", FunctionName="your-function-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable error volume.

Evaluation interval: 60

Pending period: 180

Recovery period: 300

Throttles

Labels: FunctionName

Alarm description: Alarm when function invocations are throttled for a Lambda function.

Intent: Detect function invocation throttling.

PromQL criteria: `sum({__name__="Throttles", "@instrumentation.@name"="cloudwatch.aws/lambda", FunctionName="`*your-function-name*`"}) > `*THRESHOLD*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Throttling indicates concurrency limit reached.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Duration

Labels: FunctionName

Alarm description: Alarm when p90 function duration exceeds the threshold for a Lambda function.

Intent: Detect long-running function invocations.

PromQL criteria: `histogram_quantile(0.9, {__name__="Duration", "@instrumentation.@name"="cloudwatch.aws/lambda", FunctionName="`*your-function-name*`"}) > `*THRESHOLD*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set relative to function timeout.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

ConcurrentExecutions

Labels: FunctionName

Alarm description: Alarm when concurrent executions exceed the threshold for a Lambda function.

Intent: Detect function approaching concurrency limits.

PromQL criteria: `max({__name__="ConcurrentExecutions", "@instrumentation.@name"="cloudwatch.aws/lambda", FunctionName="`*your-function-name*`"}) > `*THRESHOLD*

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set to percentage of reserved concurrency.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

AWS Lambda Insights

memory_utilization

Labels: function_name

Alarm description: Alarm when average memory utilization exceeds 90% for a Lambda function.

Intent: Detect function approaching configured memory limit.

PromQL criteria: `avg({__name__="memory_utilization", "@instrumentation.@name"="cloudwatch.aws/lambda-insights", function_name="your-function-name"}) > 90`

Recommended threshold: 90 (embedded in query)

Threshold justification: Above 90% risks out-of-memory failures.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

NAT Gateway

ErrorPortAllocation

Labels: NatGatewayId

Alarm description: Alarm when the NAT gateway fails to allocate a port for new connections.

Intent: Detect port allocation failures preventing new connections.

PromQL criteria: `sum({__name__="ErrorPortAllocation", "@instrumentation.@name"="cloudwatch.aws/natgateway", NatGatewayId="your-nat-gateway-id"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any allocation failure impacts connectivity.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

PacketsDropCount

Labels: NatGatewayId

Alarm description: Alarm when the NAT gateway drops packets exceeding the threshold.

Intent: Detect packet drops indicating capacity or connectivity issues.

PromQL criteria: `sum({__name__="PacketsDropCount", "@instrumentation.@name"="cloudwatch.aws/natgateway", NatGatewayId="your-nat-gateway-id"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable drop rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

AWS PrivateLink endpoints

PacketsDropped

Labels: VpcId, VpcEndpointId, EndpointType, SubnetId, ServiceName

Alarm description: Alarm when a VPC endpoint drops packets exceeding the threshold.

Intent: Detect unhealthy endpoint or endpoint service.

PromQL criteria: `sum({__name__="PacketsDropped", "@instrumentation.@name"="cloudwatch.aws/privatelinkendpoints", VpcId="your-vpc-id",VpcEndpointId="your-vpc-endpoint-id",EndpointType="your-endpoint-type",SubnetId="your-subnet-id",ServiceName="your-service-name"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable drop rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

AWS PrivateLink services

RstPacketsSent

Labels: ServiceId, LoadBalancerArn, Az

Alarm description: Alarm when the RST packet rate exceeds the threshold for an endpoint service.

Intent: Detect unhealthy targets of endpoint service.

PromQL criteria: `sum({__name__="RstPacketsSent", "@instrumentation.@name"="cloudwatch.aws/privatelinkservices", ServiceId="SVC",LoadBalancerArn="LB",Az="AZ"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable RST packet rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

RDS

CPUUtilization

Labels: DBInstanceIdentifier

Alarm description: Alarm when average CPU utilization exceeds 90% for an RDS instance.

Intent: Detect high CPU preventing performance degradation.

PromQL criteria: `avg({__name__="CPUUtilization", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > 90`

Recommended threshold: 90 (embedded in query)

Threshold justification: 90% indicates near saturation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

DatabaseConnections

Labels: DBInstanceIdentifier

Alarm description: Alarm when database connections exceed the threshold for an RDS instance.

Intent: Prevent rejected connections at maximum limit.

PromQL criteria: `avg({__name__="DatabaseConnections", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set to percentage of max_connections parameter.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

EBSByteBalance%

Labels: DBInstanceIdentifier

Alarm description: Alarm when EBS byte balance drops below 10% for an RDS instance.

Intent: Detect low throughput credits causing bottlenecks.

PromQL criteria: `avg({__name__="EBSByteBalance%", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) < 10`

Recommended threshold: 10 (embedded in query)

Threshold justification: Below 10% risks throughput degradation.

Evaluation interval: 60

Pending period: 180

Recovery period: 180

EBSIOBalance%

Labels: DBInstanceIdentifier

Alarm description: Alarm when EBS IO balance drops below 10% for an RDS instance.

Intent: Detect low IOPS credits causing bottlenecks.

PromQL criteria:
```
avg({__name__="EBSIOBalance%",
"@instrumentation.@name"="cloudwatch.aws/rds",
DBInstanceIdentifier="DB"}) < 10
```

Recommended threshold: 10 (embedded in query)

Threshold justification: Below 10% risks IOPS degradation.

Evaluation interval: 60

Pending period: 180

Recovery period: 180

FreeableMemory

Labels: DBInstanceIdentifier

Alarm description: Alarm when freeable memory drops below the threshold for an RDS instance.

Intent: Prevent out-of-memory causing rejected connections.

PromQL criteria:
```
avg({__name__="FreeableMemory",
"@instrumentation.@name"="cloudwatch.aws/rds",
DBInstanceIdentifier="DB"}) < THRESHOLD
```

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on instance class memory.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

FreeLocalStorage

Labels: DBInstanceIdentifier

Alarm description: Alarm when free local storage drops below the threshold for an Aurora instance.

Intent: Prevent Aurora local storage exhaustion.

PromQL criteria: `avg({__name__="FreeLocalStorage", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on minimum required for operations.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

FreeStorageSpace

Labels: DBInstanceIdentifier

Alarm description: Alarm when free storage space drops below the threshold for an RDS instance.

Intent: Prevent storage-full downtime.

PromQL criteria: `min({__name__="FreeStorageSpace", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on storage growth rate and provisioned size.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

MaximumUsedTransactionIDs

Labels: DBInstanceIdentifier

Alarm description: Alarm when maximum used transaction IDs exceed 1 billion for an RDS instance.

Intent: Prevent PostgreSQL transaction ID wraparound.

PromQL criteria: `avg({__name__="MaximumUsedTransactionIDs", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > 1000000000`

Recommended threshold: 1000000000 (embedded in query)

Threshold justification: 1 billion indicates approaching wraparound danger.

Evaluation interval: 60

Pending period: 60

Recovery period: 60

ReadLatency

Labels: DBInstanceIdentifier

Alarm description: Alarm when p90 read latency exceeds the threshold for an RDS instance.

Intent: Detect high read latency indicating disk issues.

PromQL criteria: `histogram_quantile(0.9, {__name__="ReadLatency", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable application latency.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

ReplicaLag

Labels: DBInstanceIdentifier

Alarm description: Alarm when replication lag exceeds 60 seconds for an RDS read replica.

Intent: Detect replication delay risking data loss.

PromQL criteria:
```
max({__name__="ReplicaLag",
"@instrumentation.@name"="cloudwatch.aws/rds",
DBInstanceIdentifier="DB"}) > 60
```

Recommended threshold: 60 (embedded in query)

Threshold justification: 60 seconds represents significant lag for most workloads.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

WriteLatency

Labels: DBInstanceIdentifier

Alarm description: Alarm when p90 write latency exceeds the threshold for an RDS instance.

Intent: Detect high write latency indicating disk issues.

PromQL criteria:
```
histogram_quantile(0.9, {__name__="WriteLatency",
"@instrumentation.@name"="cloudwatch.aws/rds",
DBInstanceIdentifier="DB"}) > THRESHOLD
```

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable application latency.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

DBLoad

Labels: DBInstanceIdentifier

Alarm description: Alarm when average database load exceeds the threshold for an RDS instance.

Intent: Detect high database load degrading performance.

PromQL criteria: `avg({__name__="DBLoad", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set to multiple of vCPU count.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

AuroraVolumeBytesLeftTotal

Labels: DBClusterIdentifier

Alarm description: Alarm when Aurora cluster storage bytes remaining drops below the threshold.

Intent: Prevent Aurora cluster storage exhaustion.

PromQL criteria: `avg({__name__="AuroraVolumeBytesLeftTotal", "@instrumentation.@name"="cloudwatch.aws/rds", DBClusterIdentifier="CLUSTER"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on growth rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

AuroraBinlogReplicaLag

Labels: DBClusterIdentifier

Alarm description: Alarm when Aurora binlog replica lag indicates an error state.

Intent: Detect writer instance replication errors.

PromQL criteria: `avg({__name__="AuroraBinlogReplicaLag", "@instrumentation.@name"="cloudwatch.aws/rds", DBClusterIdentifier="CLUSTER"}) <= -1`

Recommended threshold: -1 (embedded in query)

Threshold justification: -1 indicates error state.

Evaluation interval: 60

Pending period: 120

Recovery period: 120

BlockedTransactions

Labels: DBInstanceIdentifier

Alarm description: Alarm when blocked transaction count exceeds the threshold for an RDS instance.

Intent: Detect transaction blocking causing performance degradation.

PromQL criteria: `avg({__name__="BlockedTransactions", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable blocked transaction count.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

BufferCacheHitRatio

Labels: DBInstanceIdentifier

Alarm description: Alarm when buffer cache hit ratio drops below 80% for an RDS instance.

Intent: Detect low cache efficiency causing performance decrease.

PromQL criteria: `avg({__name__="BufferCacheHitRatio", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) < 80`

Recommended threshold: 80 (embedded in query)

Threshold justification: Below 80% indicates excessive disk I/O.

Evaluation interval: 60

Pending period: 600

Recovery period: 600

EngineUptime

Labels: DBClusterIdentifier

Alarm description: Alarm when engine uptime drops to zero indicating an Aurora writer restart.

Intent: Detect Aurora writer instance downtime or crash.

PromQL criteria: `avg({__name__="EngineUptime", "@instrumentation.@name"="cloudwatch.aws/rds", DBClusterIdentifier="CLUSTER"}) <= 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Zero uptime indicates instance restart.

Evaluation interval: 60

Pending period: 120

Recovery period: 120

RollbackSegmentHistoryListLength

Labels: DBInstanceIdentifier

Alarm description: Alarm when rollback segment history list length exceeds 1 million for an Aurora instance.

Intent: Detect high rollback history causing CPU degradation.

PromQL criteria: `avg({__name__="RollbackSegmentHistoryListLength", "@instrumentation.@name"="cloudwatch.aws/rds", DBInstanceIdentifier="DB"}) > 1000000`

Recommended threshold: 1000000 (embedded in query)

Threshold justification: Above 1M causes performance issues.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

StorageNetworkThroughput

Labels: DBClusterIdentifier

Alarm description: Alarm when storage network throughput exceeds the threshold for an Aurora cluster.

Intent: Detect high throughput risking network packet drops.

PromQL criteria: `avg({__name__="StorageNetworkThroughput", "@instrumentation.@name"="cloudwatch.aws/rds", DBClusterIdentifier="CLUSTER"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on instance network capacity.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

Route 53

HealthCheckStatus

Labels: HealthCheckId

Alarm description: Alarm when a Route 53 health check reports an unhealthy endpoint.

Intent: Detect unhealthy endpoints by using Route 53 health checkers.

PromQL criteria: `avg({__name__="HealthCheckStatus", "@instrumentation.@name"="cloudwatch.aws/route53", HealthCheckId="HC"}) < 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: Below 1 indicates endpoint is failing health checks.

Evaluation interval: 60

Pending period: 180

Recovery period: 180

S3

4xxErrors

Labels: BucketName, FilterId

Alarm description: Alarm when the 4xx error rate exceeds 5% for an S3 bucket.

Intent: Detect abnormal client error rates.

PromQL criteria: `avg({__name__="4xxErrors", "@instrumentation.@name"="cloudwatch.aws/s3", BucketName="BUCKET",FilterId="FILTER"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Above 5% indicates setup or access issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

5xxErrors

Labels: BucketName, FilterId

Alarm description: Alarm when the 5xx error rate exceeds 5% for an S3 bucket.

Intent: Detect server-side errors affecting application.

PromQL criteria: `avg({__name__="5xxErrors", "@instrumentation.@name"="cloudwatch.aws/s3", BucketName="BUCKET",FilterId="FILTER"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Above 5% indicates S3 service issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

OperationsFailedReplication

Labels: SourceBucket, DestinationBucket, RuleId

Alarm description: Alarm when S3 replication operations fail for a bucket.

Intent: Detect replication failures risking data inconsistency.

PromQL criteria: `max({__name__="OperationsFailedReplication", "@instrumentation.@name"="cloudwatch.aws/s3", SourceBucket="SRC",DestinationBucket="DST",RuleId="RULE"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any failed replication requires investigation.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

S3 Object Lambda

4xxErrors

Labels: AccessPointName, DataSourceARN

Alarm description: Alarm when the 4xx error rate exceeds 5% for an S3 Object Lambda access point.

Intent: Detect abnormal client error rates for Object Lambda.

PromQL criteria: `avg({__name__="4xxErrors", "@instrumentation.@name"="cloudwatch.aws/s3objectlambda", AccessPointName="AP",DataSourceARN="ARN"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Above 5% indicates setup issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

5xxErrors

Labels: AccessPointName, DataSourceARN

Alarm description: Alarm when the 5xx error rate exceeds 5% for an S3 Object Lambda access point.

Intent: Detect server-side errors in Object Lambda.

PromQL criteria: `avg({__name__="5xxErrors", "@instrumentation.@name"="cloudwatch.aws/s3objectlambda", AccessPointName="AP",DataSourceARN="ARN"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Above 5% indicates Lambda or S3 issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

LambdaResponse4xx

Labels: AccessPointName, DataSourceARN

Alarm description: Alarm when the Lambda function 4xx response rate exceeds 5% for an S3 Object Lambda access point.

Intent: Detect Lambda function client errors.

PromQL criteria: `avg({__name__="LambdaResponse4xx", "@instrumentation.@name"="cloudwatch.aws/s3objectlambda", AccessPointName="AP",DataSourceARN="ARN"}) > 0.05`

Recommended threshold: 0.05 (embedded in query)

Threshold justification: Above 5% indicates Lambda function issues.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

SNS

NumberOfMessagesPublished

Labels: TopicName

Alarm description: Alarm when the number of messages published drops below the threshold for an SNS topic.

Intent: Detect significant drop in publishing volume.

PromQL criteria: `sum({__name__="NumberOfMessagesPublished", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on minimum expected traffic.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsDelivered

Labels: TopicName

Alarm description: Alarm when the number of notifications delivered drops below the threshold for an SNS topic.

Intent: Detect drop in notification delivery volume.

PromQL criteria: `sum({__name__="NumberOfNotificationsDelivered", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on minimum expected deliveries.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsFailed

Labels: TopicName

Alarm description: Alarm when failed notification delivery count exceeds the threshold for an SNS topic.

Intent: Detect delivery failures.

PromQL criteria: `sum({__name__="NumberOfNotificationsFailed", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable failure rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsFilteredOut-InvalidAttributes

Labels: TopicName

Alarm description: Alarm when notifications are filtered out due to invalid message attributes.

Intent: Detect invalid message attributes.

PromQL criteria: `sum({__name__="NumberOfNotificationsFilteredOut-InvalidAttributes", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any invalid filter indicates misconfiguration.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsFilteredOut-InvalidMessageBody

Labels: TopicName

Alarm description: Alarm when notifications are filtered out due to invalid message bodies.

Intent: Detect invalid message bodies.

PromQL criteria: `sum({__name__="NumberOfNotificationsFilteredOut-InvalidMessageBody", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any invalid filter indicates misconfiguration.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsRedrivenToDlq

Labels: TopicName

Alarm description: Alarm when notifications are sent to the dead-letter queue for an SNS topic.

Intent: Detect messages sent to dead-letter queue.

PromQL criteria: `sum({__name__="NumberOfNotificationsRedrivenToDlq", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Any DLQ message indicates delivery issues.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

NumberOfNotificationsFailedToRedriveToDlq

Labels: TopicName

Alarm description: Alarm when notifications fail to be sent to the dead-letter queue for an SNS topic.

Intent: Detect DLQ delivery failures.

PromQL criteria: `sum({__name__="NumberOfNotificationsFailedToRedriveToDlq", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Failed DLQ means lost error tracking.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

SMSMonthToDateSpentUSD

Labels: TopicName

Alarm description: Alarm when SMS spending approaches the account quota for an SNS topic.

Intent: Detect SMS spend approaching quota.

PromQL criteria: `max({__name__="SMSMonthToDateSpentUSD", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on account SMS quota.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

SMSSuccessRate

Labels: TopicName

Alarm description: Alarm when the SMS delivery success rate drops below the threshold for an SNS topic.

Intent: Detect failing SMS deliveries.

PromQL criteria: `avg({__name__="SMSSuccessRate", "@instrumentation.@name"="cloudwatch.aws/sns", TopicName="TOPIC"}) < THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable delivery rate.

Evaluation interval: 60

Pending period: 300

Recovery period: 300

SQS

ApproximateAgeOfOldestMessage

Labels: QueueName

Alarm description: Alarm when the oldest message age exceeds the threshold for an SQS queue.

Intent: Detect messages not processed quickly enough.

PromQL criteria: `max({__name__="ApproximateAgeOfOldestMessage", "@instrumentation.@name"="cloudwatch.aws/sqs", QueueName="QUEUE"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on acceptable message processing time.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

ApproximateNumberOfMessagesNotVisible

Labels: QueueName

Alarm description: Alarm when the number of in-flight messages exceeds the threshold for an SQS queue.

Intent: Detect high in-flight messages indicating consumer issues.

PromQL criteria: `avg({__name__="ApproximateNumberOfMessagesNotVisible", "@instrumentation.@name"="cloudwatch.aws/sqs", QueueName="QUEUE"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on expected processing volume.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

ApproximateNumberOfMessagesVisible

Labels: QueueName

Alarm description: Alarm when the number of visible messages exceeds the threshold for an SQS queue.

Intent: Detect message backlog indicating slow consumers.

PromQL criteria: `avg({__name__="ApproximateNumberOfMessagesVisible", "@instrumentation.@name"="cloudwatch.aws/sqs", QueueName="QUEUE"}) > THRESHOLD`

Recommended threshold: *THRESHOLD* (embedded in query)

Threshold justification: Set based on expected queue depth.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

NumberOfMessagesSent

Labels: QueueName

Alarm description: Alarm when no messages are sent to an SQS queue.

Intent: Detect producers stopped sending messages.

PromQL criteria: `sum({__name__="NumberOfMessagesSent", "@instrumentation.@name"="cloudwatch.aws/sqs", QueueName="QUEUE"}) <= 0`

Recommended threshold: 0 (embedded in query)

Threshold justification: Zero messages indicates producer failure.

Evaluation interval: 60

Pending period: 900

Recovery period: 900

VPN

TunnelState (VPN level)

Labels: VpnId

Alarm description: Alarm when at least one VPN tunnel is in a DOWN state.

Intent: Detect at least one tunnel in DOWN state.

PromQL criteria: `min({__name__="TunnelState", "@instrumentation.@name"="cloudwatch.aws/vpn", VpnId="VPN"}) < 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: Below 1 means tunnel is down.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

TunnelState (Tunnel level)

Labels: TunnelIpAddress

Alarm description: Alarm when a specific VPN tunnel is in a DOWN state.

Intent: Detect specific tunnel in DOWN state.

PromQL criteria: `min({__name__="TunnelState", "@instrumentation.@name"="cloudwatch.aws/vpn", TunnelIpAddress="IP"}) < 1`

Recommended threshold: 1 (embedded in query)

Threshold justification: Below 1 means tunnel is down.

Evaluation interval: 300

Pending period: 900

Recovery period: 900

Recommended alarms (Classic)

The following sections list the metrics that we recommend that you set best practice alarms for. For each metric, the dimensions, alarm intent, recommended threshold, threshold justification, and the period length and number of datapoints is also displayed.

Some metrics might appear twice in the list. This happens when different alarms are recommended for different combinations of dimensions of that metric.

Datapoints to alarm is the number of data points that must be breaching to send the alarm into ALARM state. **Evaluation periods** is the number of periods that are taken into account when the alarm is evaluated. If these numbers are the same, the alarm goes into ALARM state only when that number of consecutive periods have values that breach the threshold. If **Datapoints to alarm** is lower than **Evaluation periods**, then it is an "M out of N" alarm and the alarm goes into ALARM state if at least **Datapoints to alarm** data points are breaching within any **Evaluation periods** set of data points. For more information, see Alarm evaluation.

Topics

- Amazon API Gateway
- Amazon EC2 Auto Scaling
- AWS Certificate Manager (ACM)
- Amazon CloudFront
- Amazon Cognito
- Amazon DynamoDB

- Amazon EBS
- Amazon EC2
- Amazon ElastiCache
- Amazon ECS
- Amazon ECS with Container Insights
- Amazon ECS with Container Insights with enhanced observability
- Amazon EFS
- Amazon EKS with Container Insights
- Amazon EventBridge Scheduler
- Amazon Kinesis Data Streams
- Lambda
- Lambda Insights
- Amazon VPC (AWS/NATGateway)
- AWS Private Link (AWS/PrivateLinkEndpoints)
- AWS Private Link (AWS/PrivateLinkServices)
- Amazon RDS
- Amazon Route 53 Public Data Plane
- Amazon S3
- S3ObjectLambda
- Amazon SNS
- Amazon SQS
- Site-to-Site VPN

Amazon API Gateway

4XXError

Dimensions: ApiName, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a resource was removed or a client is requesting one that doesn't exist. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4XX errors. Moreover, consider enabling

detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4XX errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5XXError

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the API gateway and the backend API. This documentation can help you troubleshoot the cause of 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests as

well as acceptable error rates. you can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage

Alarm description: This alarm helps to detect low traffic volume for the REST API stage. This can be an indicator of an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of traffic volume drops for each resource and method. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the REST API resource and method in the stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the REST API resource and method in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect low traffic volume for the HTTP API stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It

could also be an indicator of an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, we recommend that you create alternative alarms to this in order to have more fine-grained monitoring of traffic volume drops for each route. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

Count

Dimensions: ApiId, Stage, Resource, Method

Alarm description: This alarm helps to detect low traffic volume for the HTTP API route in the stage. This can indicate an issue with the application calling the API such as using incorrect endpoints. It could also indicate an issue with the configuration or permissions of the API making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the HTTP API route in the stage. We recommend that you create this alarm if your API receives a predictable and consistent number of requests under normal conditions. This alarm is not recommended for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline request count for your API is. Setting the threshold at a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it at a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

IntegrationLatency

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect if there is high integration latency for the API requests in a stage. You can correlate the `IntegrationLatency` metric value with the corresponding latency metric of your backend such as the `Duration` metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues, or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the integration latency.

Intent: This alarm can detect when the API Gateway requests in a stage have a high integration latency. We recommend this alarm for WebSocket APIs, and it is considered optional for HTTP APIs because they already have separate alarm recommendations for the Latency metric. If you have detailed CloudWatch metrics enabled and you have different integration latency performance requirements per route, we recommend that you create alternative alarms in order to have more fine-grained monitoring of the integration latency for each route.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If is acceptable for the API to have a higher latency in general, set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

IntegrationLatency

Dimensions: ApiId, Stage, Route

Alarm description: This alarm helps to detect if there is high integration latency for the WebSocket API requests for a route in a stage. You can correlate the `IntegrationLatency` metric value with the corresponding latency metric of your backend such as the `Duration` metric for Lambda integrations. This helps you determine whether the API backend is taking more time to process requests from clients due to performance issues or if there is some other overhead from initialization or cold start. Additionally, consider enabling CloudWatch Logs for your API and checking the logs for any errors that may be causing the high latency issues.

Intent: This alarm can detect when the API Gateway requests for a route in a stage have high integration latency.

Statistic: p90

Recommended threshold: 2000.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is

expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload, and then used to tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage

Alarm description: This alarm detects high latency in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for issues. Consider also enabling CloudWatch Logs and checking for errors that might be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per resource and method and narrow down the source of the latency. If applicable, refer to the troubleshooting with Lambda or troubleshooting for edge-optimized API endpoints guides.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements for each method and resource, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each resource and method.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine what the expected baseline latency is for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiName, Stage, Resource, Method

Alarm description: This alarm detects high latency for a resource and method in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that might be causing the high latency. You can also refer to the troubleshooting with Lambda or troubleshooting for edge-optimized API endpoints guides if applicable.

Intent: This alarm can detect when the API Gateway requests for a resource and method in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, you can use it as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiId, Stage

Alarm description: This alarm detects high latency in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and you should investigate there for performance issues. Consider also enabling CloudWatch Logs and checking for any errors that may be causing the high latency. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route and narrow down the source of the latency. You can also refer to the troubleshooting with Lambda integrations guide if applicable.

Intent: This alarm can detect when the API Gateway requests in a stage have high latency. If you have detailed CloudWatch metrics enabled and you have different latency performance requirements per route, we recommend that you create alternative alarms to have more fine-grained monitoring of the latency for each route.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make it less sensitive.However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Latency

Dimensions: ApiId, Stage, Resource, Method

Alarm description: This alarm detects high latency for a route in a stage. Find the `IntegrationLatency` metric value to check the API backend latency. If the two metrics are mostly aligned, the API backend is the source of higher latency and should be investigated for performance issues. Consider also enabling CloudWatch logs and checking for any errors that might be causing the high latency. You can also refer to the troubleshooting with Lambda integrations guide if applicable.

Intent: This alarm is used to detect when the API Gateway requests for a route in a stage have high latency.

Statistic: p90

Recommended threshold: 2500.0

Threshold justification: The suggested threshold value does not work for all the API workloads. However, it can be used as a starting point for the threshold. You can then choose different threshold values based on the workload and acceptable latency, performance, and SLA requirements for the API. If it is acceptable for the API to have a higher latency in general, you can set a higher threshold value to make the alarm less sensitive. However, if the API is expected to provide near real-time responses, set a lower threshold value. You can also analyze historical data to determine the expected baseline latency for the application workload and then tune the threshold value accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

4xx

Dimensions: ApiId, Stage

Alarm description: This alarm detects a high rate of client-side errors. This can indicate an issue in the authorization or client request parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors can also be caused by exceeding the configured throttling limit. If the

responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

5xx

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect a high rate of server-side errors. This can indicate that there is something wrong on the API backend, the network, or the integration between the API gateway and the backend API. This documentation can help you troubleshoot the cause for 5xx errors.

Intent: This alarm can detect high rates of server-side errors for the API Gateway requests.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 5xx errors. However, you can tune the threshold to suit the traffic of the requests as well as acceptable error rates. You can also analyze historical data to determine what the

acceptable error rate is for the application workload, and then you can tune the threshold accordingly. Frequently occurring 5xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

MessageCount

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect low traffic volume for the WebSocket API stage. This can indicate an issue when clients call the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. If you have detailed CloudWatch metrics enabled and you can predict the normal traffic volume per route, it is better to create alternative alarms to this one, in order to have more fine-grained monitoring of traffic volume drops for each route. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold value based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

MessageCount

Dimensions: ApiId, Stage, Route

Alarm description: This alarm helps detect low traffic volume for the WebSocket API route in the stage. This can indicate an issue with the clients calling the API such as using incorrect endpoints, or issues with the backend sending messages to clients. It could also indicate an issue with the configuration or permissions of the API, making it unreachable for clients.

Intent: This alarm can detect unexpectedly low traffic volume for the WebSocket API route in the stage. We recommend that you create this alarm if your API receives and sends a predictable and consistent number of messages under normal conditions. We do not recommend this alarm for APIs that don't expect constant and consistent traffic.

Statistic: SampleCount

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold based on historical data analysis to determine what the expected baseline message count for your API is. Setting the threshold to a very high value might cause the alarm to be too sensitive at periods of normal and expected low traffic. Conversely, setting it to a very low value might cause the alarm to miss anomalous smaller drops in traffic volume.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

ClientError

Dimensions: ApiId, Stage

Alarm description: This alarm detects a high rate of client errors. This can indicate an issue in the authorization or message parameters. It could also mean that a route was removed or a client is requesting one that doesn't exist in the API. Consider enabling CloudWatch Logs

and checking for any errors that may be causing the 4xx errors. Moreover, consider enabling detailed CloudWatch metrics to view this metric per route, to help you narrow down the source of the errors. Errors could also be caused by exceeding the configured throttling limit. If the responses and logs are reporting high and unexpected rates of 429 errors, follow this guide to troubleshoot this issue.

Intent: This alarm can detect high rates of client errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting 4xx errors. You can tune the threshold to suit the traffic of the requests as well as to suit your acceptable error rates. You can also analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring 4xx errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ExecutionError

Dimensions: ApiId, Stage

Alarm description: This alarm helps to detect a high rate of execution errors. This can be caused by 5xx errors from your integration, permission issues, or other factors preventing successful invocation of the integration, such as the integration being throttled or deleted. Consider enabling CloudWatch Logs for your API and checking the logs for the type and cause of the errors. Moreover, consider enabling detailed CloudWatch metrics to get a view of this metric per route, to help you narrow down the source of the errors. This documentation can also help you troubleshoot the cause of any connection errors.

Intent: This alarm can detect high rates of execution errors for the WebSocket API Gateway messages.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The suggested threshold detects when more than 5% of total requests are getting execution errors. You can tune the threshold to suit the traffic of the requests, as well as to suit your acceptable error rates. You can analyze historical data to determine the acceptable error rate for the application workload, and then tune the threshold accordingly. Frequently occurring execution errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EC2 Auto Scaling

GroupInServiceCapacity

Dimensions: AutoScalingGroupName

Alarm description: This alarm helps to detect when the capacity in the group is below the desired capacity required for your workload. To troubleshoot, check your scaling activities for launch failures and confirm that your desired capacity configuration is correct.

Intent: This alarm can detect a low availability in your auto scaling group because of launch failures or suspended launches.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The threshold value should be the minimum capacity required to run your workload. In most cases, you can set this to match the GroupDesiredCapacity metric.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

AWS Certificate Manager (ACM)

DaysToExpiry

Dimensions: CertificateArn

Alarm description: This alarm helps you detect when a certificate managed by or imported into ACM is approaching its expiration date. It helps to prevent unexpected certificate expirations that could lead to service disruptions. When the alarm transitions into ALARM state, you should take immediate action to renew or re-import the certificate. For ACM-managed certificates, see the instructions at certificate renewal process. For imported certificates, see the instructions at re-import process.

Intent: This alarm can proactively alert you about upcoming certificate expirations. It provides sufficient advance notice to allow for manual intervention, enabling you to renew or replace certificates before they expire. This helps you maintain the security and availability of TLS-enabled services. When this goes into ALARM, immediately investigate the certificate status and initiate the renewal process if necessary.

Statistic: Minimum

Recommended threshold: 44.0

Threshold justification: The 44-day threshold provides a balance between early warning and avoiding false alarms. It allows sufficient time for manual intervention if automatic renewal fails. Adjust this value based on your certificate renewal process and operational response times.

Period: 86400

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon CloudFront

5xxErrorRate

Dimensions: DistributionId, Region=Global

Alarm description: This alarm monitors the percentage of 5xx error responses from your origin server, to help you detect if the CloudFront service is having issues. See Troubleshooting error responses from your origin for information to help you understand the problems with your server. Also, turn on additional metrics to get detailed error metrics.

Intent: This alarm is used to detect problems with serving requests from the origin server, or problems with communication between CloudFront and your origin server.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the tolerance for 5xx responses. You can analyze historical data and trends, and then set the threshold accordingly. Because 5xx errors can be caused by transient issues, we recommend that you set the threshold to a value greater than 0 so that the alarm is not too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

OriginLatency

Dimensions: DistributionId, Region=Global

Alarm description: The alarm helps to monitor if the origin server is taking too long to respond. If the server takes too long to respond, it might lead to a timeout. Refer to find and fix delayed responses from applications on your origin server if you experience consistently high `OriginLatency` values.

Intent: This alarm is used to detect problems with the origin server taking too long to respond.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 80% of the origin response timeout, and use the result as the threshold value. If this metric is consistently close to the origin response timeout value, you might start experiencing 504 errors.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionValidationErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor validation errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem. See Restrictions on edge functions to understand the common misconfigurations for CloudFront Functions.

Intent: This alarm is used to detect validation errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: A value greater than 0 indicates a validation error. We recommend setting the threshold to 0 because validation errors imply a problem when CloudFront functions hand off back to CloudFront. For example, CloudFront needs the HTTP Host header to process a request. There is nothing stopping a user from deleting the Host header in their CloudFront functions code. But when CloudFront gets the response back and the Host header is missing, CloudFront throws a validation error.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionExecutionErrors

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you monitor execution errors from CloudFront functions so that you can take steps to resolve them. Analyze the CloudWatch function logs and look at the function code to find and resolve the root cause of the problem.

Intent: This alarm is used to detect execution errors from CloudFront functions.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend to set the threshold to 0 because an execution error indicates a problem with the code that occurs at runtime.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FunctionThrottles

Dimensions: DistributionId, FunctionName, Region=Global

Alarm description: This alarm helps you to monitor if your CloudFront function is throttled. If your function is throttled, it means that it is taking too long to execute. To avoid function throttles, consider optimizing the function code.

Intent: This alarm can detect when your CloudFront function is throttled so that you can react and resolve the issue for a smooth customer experience.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: We recommend setting the threshold to 0, to allow quicker resolution of the function throttles.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Cognito

SignUpThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled requests. If users are consistently getting throttled, you should increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase. To take actions proactively, consider tracking the usage quota.

Intent: This alarm helps to monitor the occurrence of throttled sign-up requests. This can help you know when to take actions to mitigate any degradation in sign-up experience. Sustained throttling of requests is a negative user sign-up experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SignInThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: This alarm monitors the count of throttled user authentication requests. If users are consistently getting throttled, you might need to increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase. To take actions proactively, consider tracking the usage quota.

Intent: This alarm helps to monitor the occurrence of throttled sign-in requests. This can help you know when to take actions to mitigate any degradation in sign-in experience. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: A well-provisioned user pool should not encounter any throttling which spans across multiple data points. So, a typical threshold for an expected workload should be zero. For an irregular workload with frequent bursts, you can analyze historical data to determine the acceptable throttling for the application workload, and then you can tune the threshold accordingly. A throttled request should be retried to minimize the impact on the application.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TokenRefreshThrottles

Dimensions: UserPool, UserPoolClient

Alarm description: You can set the threshold value can to suit the traffic of the request as well as to match acceptable throttling for token refresh requests. Throttling is used to protect your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then you can tune your alarm threshold to be higher than your acceptable throttling level. Throttled requests should be retried by the

application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Intent: This alarm helps to monitor the occurrence of throttled token refresh requests. This can help you know when to take actions to mitigate any potential problems, to make sure a smooth user experience and the health and reliability of your authentication system. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Threshold value can also be set/tuned to suit the traffic of the request as well as acceptable throttling for token refresh requests. Throttling are there for protecting your system from too many requests, however it is important to monitor if you are under provisioned for your normal traffic as well and see if it is causing the impact. Historical data can also be analyzed to see what is the acceptable throttling for the application workload and threshold can be tuned higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

FederationThrottles

Dimensions: UserPool, UserPoolClient, IdentityProvider

Alarm description: This alarm monitors the count of throttled identity federation requests. If you consistently see throttling, it might indicate that you need to increase the limit by requesting a service quota increase. Refer to Quotas in Amazon Cognito to learn how to request a quota increase.

Intent: This alarm helps to monitor the occurrence of throttled identity federation requests. This can help you take proactive responses to performance bottlenecks or misconfigurations, and make sure a smooth authentication experience for your users. Sustained throttling of requests is a bad user authentication experience.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You can set the threshold to suit the traffic of the request as well as to match the acceptable throttling for identity federation requests. Throttling is used for protecting your system from too many requests. However, it is important to monitor if you are under provisioned for your normal traffic as well. You can analyze historical data to find the acceptable throttling for the application workload, and then set the threshold to a value above your acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value for the threshold can cause alarm to be sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon DynamoDB

AccountProvisionedReadCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's read capacity is reaching its provisioned limit. You can raise the account quota for read capacity utilization if this occurs. You can view your current quotas for read capacity units and request increases using Service Quotas.

Intent: The alarm can detect if the account's read capacity utilization is approaching its provisioned read capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle read requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AccountProvisionedWriteCapacityUtilization

Dimensions: None

Alarm description: This alarm detects if the account's write capacity is reaching its provisioned limit. You can raise the account quota for write capacity utilization if this occurs. You can view your current quotas for write capacity units and request increases using Service Quotas.

Intent: This alarm can detect if the account's write capacity utilization is approaching its provisioned write capacity utilization. If the utilization reaches its maximum limit, DynamoDB starts to throttle write requests.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: Set the threshold to 80%, so that the action (such as raising the account limits) can be taken before it reaches full capacity to avoid throttling.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_THRESHOLD

AgeOfOldestUnreplicatedRecord

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the delay in replication to a Kinesis data stream. Under normal operation, `AgeOfOldestUnreplicatedRecord` should be only milliseconds. This number grows based on unsuccessful replication attempts caused by customer-controlled configuration choices. Customer-controlled configuration examples that lead to unsuccessful replication attempts are an under-provisioned Kinesis data stream capacity that leads to

excessive throttling. or a manual update to the Kinesis data stream's access policies that prevents DynamoDB from adding data to the data stream. To keep this metric as low as possible, you need to make sure the right provisioning of Kinesis data stream capacity and make sure that DynamoDB's permissions are unchanged.

Intent: This alarm can monitor unsuccessful replication attempts and the resulting delay in replication to the Kinesis data stream.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the desired replication delay measured in milliseconds. This value depends on your workload's requirements and expected performance.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

FailedToReplicateRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the number of records that DynamoDB failed to replicate to your Kinesis data stream. Certain items larger than 34 KB might expand in size to change data records that are larger than the 1 MB item size limit of Kinesis Data Streams. This size expansion occurs when these larger than 34 KB items include a large number of Boolean or empty attribute values. Boolean and empty attribute values are stored as 1 byte in DynamoDB, but expand up to 5 bytes when they're serialized using standard JSON for Kinesis Data Streams replication. DynamoDB can't replicate such change records to your Kinesis data stream. DynamoDB skips these change data records, and automatically continues replicating subsequent records.

Intent: This alarm can monitor the number of records that DynamoDB failed to replicate to your Kinesis data stream because of the item size limit of Kinesis Data Streams.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect any records that DynamoDB failed to replicate.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName

Alarm description: This alarm detects if there are high number of read requests getting throttled for the DynamoDB table. To troubleshoot the issue, see Troubleshooting throttling issues in Amazon DynamoDB.

Intent: This alarm can detect sustained throttling for read requests to the DynamoDB table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor whether you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of read requests getting throttled for the Global Secondary Index of the DynamoDB table. To troubleshoot the issue, see Troubleshooting throttling issues in Amazon DynamoDB.

Intent: The alarm can detect sustained throttling for read requests for the Global Secondary Index of the DynamoDB Table. Sustained throttling of read requests can negatively impact your workload read operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected read traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find an acceptable throttling level for the application workload, and then tune the threshold to be higher than your usual acceptable throttling level. Throttled requests should be retried by the application or service as they are transient. Therefore, a very low threshold may cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLatency

Dimensions: TableName, ReceivingRegion

Alarm description: The alarm detects if the replica in a Region for the global table is lagging behind the source Region. The latency can increase if an AWS Region becomes degraded and you have a replica table in that Region. In this case, you can temporarily redirect your application's read and write activity to a different AWS Region. If you are using 2017.11.29 (Legacy) of global tables, you should verify that write capacity units (WCUs) are identical for

each of the replica tables. You can also make sure to follow recommendations in Best practices and requirements for managing capacity.

Intent: The alarm can detect if the replica table in a Region is falling behind replicating the changes from another Region. This could cause your replica to diverge from the other replicas. It's useful to know the replication latency of each AWS Region and alert if that replication latency increases continually. The replication of the table applies to global tables only.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Replication latencies longer than 3 minutes are generally a cause for investigation. Review the criticality and requirements of replication delay and analyze historical trends, and then select the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

SuccessfulRequestLatency

Dimensions: TableName, Operation

Alarm description: This alarm detects a high latency for the DynamoDB table operation (indicated by the dimension value of the `Operation` in the alarm). See this troubleshooting document for troubleshooting latency issues in Amazon DynamoDB.

Intent: This alarm can detect a high latency for the DynamoDB table operation. Higher latency for the operations can negatively impact the overall efficiency of the system.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: DynamoDB provides single-digit millisecond latency on average for singleton operations such as GetItem, PutItem, and so on. However, you can set the threshold based on acceptable tolerance for the latency for the type of operation and table involved in

the workload. You can analyze historical data of this metric to find the usual latency for the table operation, and then set the threshold to a number which represents critical delay for the operation.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

SystemErrors

Dimensions: TableName

Alarm description: This alarm detects a sustained high number of system errors for the DynamoDB table requests. If you continue to get 5xx errors, open the AWS Service Health Dashboard to check for operational issues with the service. You can use this alarm to get notified in case there is a prolonged internal service issue from DynamoDB and it helps you correlate with the issue your client application is facing. Refer Error handling for DynamoDB for more information.

Intent: This alarm can detect sustained system errors for the DynamoDB table requests. System errors indicate internal service errors from DynamoDB and helps correlate to the issue that the client is having.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected traffic, accounting for an acceptable level of system errors. You can also analyze historical data to find the acceptable error count for the application workload, and then tune the threshold accordingly. System errors should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ThrottledPutRecordCount

Dimensions: TableName, DelegatedOperation

Alarm description: This alarm detects the records getting throttled by your Kinesis data stream during the replication of change data capture to Kinesis. This throttling happens because of insufficient Kinesis data stream capacity. If you experience excessive and regular throttling, you might need to increase the number of Kinesis stream shards proportionally to the observed write throughput of your table. To learn more about determining the size of a Kinesis data stream, see Determining the Initial Size of a Kinesis Data Stream.

Intent: This alarm can monitor the number of records that that were throttled by your Kinesis data stream because of insufficient Kinesis data stream capacity.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You might experience some throttling during exceptional usage peaks. However, throttled records should remain as low as possible to avoid higher replication latency (DynamoDB retries sending throttled records to the Kinesis data stream). Set the threshold to a number which can help you catch regular excessive throttling. You can also analyze historical data of this metric to find the acceptable throttling rates for the application workload. Tune the threshold to a value that the application can tolerate based on your use case.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

UserErrors

Dimensions: None

Alarm description: This alarm detects a sustained high number of user errors for the DynamoDB table requests. You can check client application logs during the issue time frame to see why the requests are invalid. You can check HTTP status code 400 to see the type of error

you are getting and take action accordingly. You might have to fix the application logic to create valid requests.

Intent: This alarm can detect sustained user errors for the DynamoDB table requests. User errors for requested operations mean that the client is producing invalid requests and it is failing.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to zero to detect any client side errors. Or you can set it to a higher value if you want to avoid the alarm triggering for a very lower number of errors. Decide based on your use case and traffic for the requests.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

WriteThrottleEvents

Dimensions: TableName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for the DynamoDB table. See Troubleshooting throttling issues in Amazon DynamoDB to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests to the DynamoDB table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze

historical data to find the acceptable level of throttling for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low threshold might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

WriteThrottleEvents

Dimensions: TableName, GlobalSecondaryIndexName

Alarm description: This alarm detects if there are a high number of write requests getting throttled for Global Secondary Index of the DynamoDB table. See Troubleshooting throttling issues in Amazon DynamoDB to troubleshoot the issue.

Intent: This alarm can detect sustained throttling for write requests for the Global Secondary Index of DynamoDB Table. Sustained throttling of write requests can negatively impact your workload write operations and reduce the overall efficiency of the system.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the expected Write traffic for the DynamoDB table, accounting for an acceptable level of throttling. It is important to monitor if you are under provisioned and not causing consistent throttling. You can also analyze historical data to find the acceptable throttling level for the application workload, and then tune the threshold to a value higher than your usual acceptable throttling level. Throttled requests should be retried by the application/service as they are transient. Therefore, a very low value might cause the alarm to be too sensitive, causing unwanted state transitions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EBS

VolumeStalledIOCheck

Dimensions: VolumeId, InstanceId

Alarm description: This alarm helps you monitor the IO performance of your Amazon EBS volumes. This check detects underlying issues with the Amazon EBS infrastructure, such as hardware or software issues on the storage subsystems underlying the Amazon EBS volumes, hardware issues on the physical host that impact the reachability of the Amazon EBS volumes from your Amazon EC2 instance, and can detect connectivity issues between the instance and the Amazon EBS volumes. If the Stalled IO Check fails, you can either wait for AWS to resolve the issue, or you can take action such as replacing the affected volume or stopping and restarting the instance to which the volume is attached. In most cases, when this metric fails, Amazon EBS will automatically diagnose and recover your volume within a few minutes.

Intent: This alarm can detect the status of your Amazon EBS volumes to determine when these volumes are impaired and can not complete I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EC2

CPUUtilization

Dimensions: InstanceId

Alarm description: This alarm helps to monitor the CPU utilization of an EC2 instance. Depending on the application, consistently high utilization levels might be normal. But if performance is degraded, and the application is not constrained by disk I/O, memory, or network resources, then a maxed-out CPU might indicate a resource bottleneck or application performance problems. High CPU utilization might indicate that an upgrade to a more CPU intensive instance is required. If detailed monitoring is enabled, you can change the period to 60 seconds instead of 300 seconds. For more information, see Enable or turn off detailed monitoring for your instances.

Intent: This alarm is used to detect high CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 70-80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some systems, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your system, and set the threshold accordingly.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

StatusCheckFailed

Dimensions: InstanceId

Alarm description: This alarm helps to monitor both system status checks and instance status checks. If either type of status check fails, then this alarm should be in ALARM state.

Intent: This alarm is used to detect the underlying problems with instances, including both system status check failures and instance status check failures.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 300

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

StatusCheckFailed_AttachedEBS

Dimensions: InstanceId

Alarm description: This alarm helps you monitor whether the Amazon EBS volumes attached to an instance are reachable and able to complete I/O operations. This status check detects underlying issues with the compute or Amazon EBS infrastructure such as the following:

- Hardware or software issues on the storage subsystems underlying the Amazon EBS volumes
- Hardware issues on the physical host that impact reachability of the Amazon EBS volumes
- Connectivity issues between the instance and Amazon EBS volumes

When the attached EBS status check fails, you can either wait for Amazon to resolve the issue, or you can take an action such as replacing the affected volumes or stopping and restarting the instance.

Intent: This alarm is used to detect unreachable Amazon EBS volumes attached to an instance. These can cause failures in I/O operations.

Statistic: Maximum

Recommended threshold: 1.0

Threshold justification: When a status check fails, the value of this metric is 1. The threshold is set so that whenever the status check fails, the alarm is in ALARM state.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Amazon ElastiCache

CPUUtilization

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm helps to monitor the CPU utilization for the entire ElastiCache instance, including the database engine processes and other processes running on the instance. AWS Elasticache supports two engine types: Memcached and Redis OSS. When you reach high CPU utilization on a Memcached node, you should consider scaling up your instance type or adding new cache nodes. For Redis OSS, if your main workload is from read requests, you should consider adding more read replicas to your cache cluster. If your main workload is from write requests, you should consider adding more shards to distribute the workload across more primary nodes if you're running in clustered mode, or scaling up your instance type if you're running Redis OSS in non-clustered mode.

Intent: This alarm is used to detect high CPU utilization of ElastiCache hosts. It is useful to get a broad view of the CPU usage across the entire instance, including non-engine processes.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the percentage that reflects a critical CPU utilization level for your application. For Memcached, the engine can use up to num_threads cores. For Redis OSS, the engine is largely single-threaded, but might use additional cores if available to accelerate I/O. In most cases, you can set the threshold to about 90% of your available CPU. Because Redis OSS is single-threaded, the actual threshold value should be calculated as a fraction of the node's total capacity.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CurrConnections

Dimensions: CacheClusterId, CacheNodeId

Alarm description: This alarm detects high connection count, which might indicate heavy load or performance issues. A constant increase of `CurrConnections` might lead to exhaustion of the 65,000 available connections. It may indicate that connections improperly closed on the application side and were left established on the server side. You should consider using connection pooling or idle connection timeouts to limit the number of connections made to the cluster, or for Redis OSS, consider tuning tcp-keepalive on your cluster to detect and terminate potential dead peers.

Intent: The alarm helps you identify high connection counts that could impact the performance and stability of your ElastiCache cluster.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the acceptable range of connections for your cluster. Review the capacity and the expected workload of your ElastiCache cluster and analyze the historical connection counts during regular usage to establish a baseline, and then select a threshold accordingly. Remember that each node can support up to 65,000 concurrent connections.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseMemoryUsagePercentage

Dimensions: CacheClusterId

Alarm description: This alarm helps you monitor the memory utilization of your cluster. When your `DatabaseMemoryUsagePercentage` reaches 100%, the Redis OSS maxmemory policy is triggered and evictions might occur based on the policy selected. If no object in the cache matches the eviction policy, write operations fail. Some workloads expect or rely on evictions,

but if not, you will need to increase the memory capacity of your cluster. You can scale your cluster out by adding more primary nodes, or scale it up by using a larger node type. Refer to Scaling ElastiCache for Redis OSS clusters for details.

Intent: This alarm is used to detect high memory utilization of your cluster so that you can avoid failures when writing to your cluster. It is useful to know when you'll need to scale up your cluster if your application does not expect to experience evictions.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on your application's memory requirements and the memory capacity of your ElastiCache cluster, you should set the threshold to the percentage that reflects the critical level of memory usage of the cluster. You can use historical memory usage data as reference for acceptable memory usage threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EngineCPUUtilization

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the CPU utilization of a Redis OSS engine thread within the ElastiCache instance. Common reasons for high engine CPU are long-running commands that consume high CPU, a high number of requests, an increase of new client connection requests in a short time period, and high evictions when the cache doesn't have enough memory to hold new data. You should consider Scaling ElastiCache for Redis OSS clusters by adding more nodes or scaling up your instance type.

Intent: This alarm is used to detect high CPU utilization of the Redis OSS engine thread. It is useful if you want to monitor the CPU usage of the database engine itself.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Set the threshold to a percentage that reflects the critical engine CPU utilization level for your application. You can benchmark your cluster using your application and expected workload to correlate EngineCPUUtilization and performance as a reference, and then set the threshold accordingly. In most cases, you can set the threshold to about 90% of your available CPU.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicationLag

Dimensions: CacheClusterId

Alarm description: This alarm helps to monitor the replication health of your ElastiCache cluster. A high replication lag means that the primary node or the replica can't keep up the pace of the replication. If your write activity is too high, consider scaling your cluster out by adding more primary nodes, or scaling it up by using a larger node type. Refer to Scaling ElastiCache for Redis OSS clusters for details. If your read replicas are overloaded by the amount of read requests, consider adding more read replicas.

Intent: This alarm is used to detect a delay between data updates on the primary node and their synchronization to replica node. It helps to make sure data consistency of a read replica cluster node.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to your application's requirements and the potential impact of replication lag. You should consider your application's expected write rates and network conditions for the acceptable replication lag.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS

The following are the recommended alarms for Amazon ECS.

CPUReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high CPU reservation of the ECS cluster. High CPU reservation might indicate that the cluster is running out of registered CPUs for the task. To troubleshoot, you can add more capacity, you can scale the cluster, or you can set up auto scaling.

Intent: The alarm is used to detect whether the total number of CPU units reserved by tasks on the cluster is reaching the total CPU units registered for the cluster. This helps you know when to scale up the cluster. Reaching the total CPU units for the cluster can result in running out of CPU for tasks. If you have EC2 capacity providers managed scaling turned on, or you have associated Fargate to capacity providers, then this alarm is not recommended.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold for CPU reservation to 80%. Alternatively, you can choose a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

CPUUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high CPU utilization of the ECS service. If there is no ongoing ECS deployment, a maxed-out CPU utilization might indicate a resource

bottleneck or application performance problems. To troubleshoot, you can increase the CPU limit.

Intent: This alarm is used to detect high CPU utilization for the Amazon ECS service. Consistent high CPU utilization can indicate a resource bottleneck or application performance problems.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: The service metrics for CPU utilization might exceed 100% utilization. However, we recommend that you monitor the metric for high CPU utilization to avoid impacting other services. Set the threshold to about 80-85%. We recommend that you update your task definitions to reflect actual usage to prevent future issues with other services.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EBSFilesystemUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high storage utilization of the Amazon EBS volume attached to Amazon ECS tasks. If the utilization of the Amazon EBS volume is consistently high, you can check the usage and increase the volume size for new tasks.

Intent: This alarm is used to detect high storage utilization of the Amazon EBS volumes attached to Amazon ECS tasks. Consistently high storage utilization can indicate that the Amazon EBS volume is full and it might lead to failure of the container.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: You can set the threshold for Amazon EBS file system utilization to about 80%. You can adjust this value based on the acceptable storage utilization. For a read-only snapshot volume, a high utilization might indicate that the volume is right sized. For an

active data volume, high storage utilization might indicate that the application is writing a large amount of data which may cause the container to fail if there is not enough capacity.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

MemoryReservation

Dimensions: ClusterName

Alarm description: This alarm helps you detect a high memory reservation of the Amazon ECS cluster. High memory reservation might indicate a resource bottleneck for the cluster. To troubleshoot, analyze the service task for performance to see if memory utilization of the task can be optimized. Also, you can register more memory or set up auto scaling.

Intent: The alarm is used to detect whether the total memory units reserved by tasks on the cluster is reaching the total memory units registered for the cluster. This can help you know when to scale up the cluster. Reaching the total memory units for the cluster can cause the cluster to be unable to launch new tasks. If you have EC2 capacity providers managed scaling turned on or you have associated Fargate to capacity providers, this alarm is not recommended.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold for memory reservation to 80%. You can adjust this to a lower value based on cluster characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

MemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high memory utilization of the Amazon ECS service. If there is no ongoing Amazon ECS deployment, a maxed-out memory utilization might indicate a resource bottleneck or application performance problems. To troubleshoot, you can increase the memory limit.

Intent: This alarm is used to detect high memory utilization for the Amazon ECS service. Consistent high memory utilization can indicate a resource bottleneck or application performance problems.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: The service metrics for memory utilization might exceed 100% utilization. However, we recommend that you monitor the metric for high memory utilization to avoid impacting other services. Set the threshold to about 80%. We recommend that you update your task definitions to reflect actual usage to prevent future issues with other services.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

HTTPCode_Target_5XX_Count

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high server-side error count for the ECS service. This can indicate that there are errors that cause the server to be unable to serve requests. To troubleshoot, check your application logs.

Intent: This alarm is used to detect a high server-side error count for the ECS service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Calculate the value of about 5% of the your average traffic and use this value as a starting point for the threshold. You can find the average traffic by using the `RequestCount` metric. You can also analyze historical data to determine the acceptable error

rate for the application workload, and then tune the threshold accordingly. Frequently occurring 5XX errors need to be alarmed on. However, setting a very low value for the threshold can cause the alarm to be too sensitive.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TargetResponseTime

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a high target response time for ECS service requests. This can indicate that there are problems that cause the service to be unable to serve requests in time. To troubleshoot, check the CPUUtilization metric to see if the service is running out of CPU, or check the CPU utilization of other downstream services that your service depends on.

Intent: This alarm is used to detect a high target response time for ECS service requests.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Review the criticality and requirements of the target response time of the service and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS with Container Insights

The following are the recommended alarms for Amazon ECS with Container Insights.

EphemeralStorageUtilized

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect high ephemeral storage utilized of the Fargate cluster. If ephemeral storage is consistently high, you can check ephemeral storage usage and increase the ephemeral storage.

Intent: This alarm is used to detect high ephemeral storage usage for the Fargate cluster. Consistent high ephemeral storage utilized can indicate that the disk is full and it might lead to failure of the container.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the ephemeral storage size. You can adjust this value based on your acceptable ephemeral storage utilization of the Fargate cluster. For some systems, a consistently high ephemeral storage utilized might be normal, while for others, it might lead to failure of the container.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

RunningTaskCount

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect a low running task count of the Amazon ECS service. If the running task count is too low, it can can indicate that the application can't handle the service load and it might lead to performance issues. If there is no running task, the Amazon ECS service might be unavailable or there might be deployment issues.

Intent: This alarm is used to detect whether the number of running tasks are too low. A consistent low running task count can indicate Amazon ECS service deployment or performance issues.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: You can adjust the threshold based on the minimum running task count of the Amazon ECS service. If the running task count is 0, the Amazon ECS service will be unavailable.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

instance_filesystem_utilization

Dimensions: InstanceId, ContainerInstanceId, ClusterName

Alarm description: This alarm helps you detect a high file system utilization of the Amazon ECS cluster. If the file system utilization is consistently high, check the disk usage.

Intent: This alarm is used to detect high file system utilization for the Amazon ECS cluster. A consistent high file system utilization can indicate a resource bottleneck or application performance problems, and it might prevent running new tasks.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: You can set the threshold for file system utilization to about 90-95%. You can adjust this value based on the acceptable file system capacity level of the Amazon ECS cluster. For some systems, a consistently high file system utilization might be normal and not indicate a problem, while for others, it might be a cause of concern and might lead to performance issues and prevent running new tasks.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon ECS with Container Insights with enhanced observability

The following are the recommended alarms for Amazon ECS with Container Insights with enhanced observability.

TaskCpuUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high CPU utilization of tasks in your Amazon ECS cluster. If task CPU utilization is consistently high, you might need to optimize your tasks or increase their CPU reservation.

Intent: This alarm is used to detect high CPU utilization for tasks in the Amazon ECS cluster. Consistent high CPU utilization can indicate that the tasks are under stress and might need more CPU resources or optimization to maintain performance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's CPU reservation.
You can adjust this value based on your acceptable CPU utilization for the tasks. For some workloads, consistently high CPU utilization might be normal, while for others, it might indicate performance issues or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskCpuUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of CPU units being used by a task.

Intent: This alarm is used to detect high task CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some tasks, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskMemoryUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high memory utilization of tasks in your Amazon ECS cluster. If memory utilization is consistently high, you might need to optimize your tasks or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for tasks in the Amazon ECS cluster. Consistent high memory utilization can indicate that the task is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's memory reservation. You can adjust this value based on your acceptable memory utilization for the tasks. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskMemoryUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of memory being used by a task.

Intent: This alarm is used to detect high memory utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for memory utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some tasks, consistently high memory utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical memory utilization data to identify the usage, find what memory utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

ContainerCpuUtilization

Dimensions: ClusterName

Alarm description: This alarm monitors the percentage of CPU units used by containers in your Amazon ECS cluster relative to their reserved CPU. It helps detect when containers are approaching their CPU limits based on the ContainerCpuUtilized/ContainerCpuReserved ratio.

Intent: This alarm detects when containers in the Amazon ECS cluster are using a high percentage of their reserved CPU capacity, calculated as `ContainerCpuUtilized/ContainerCpuReserved`. Sustained high values indicate containers are operating near their CPU limits and might need capacity adjustments.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's CPU utilization ratio. This provides an early warning when containers are approaching their CPU capacity limits while allowing for normal fluctuations in CPU usage. The threshold can be adjusted based on your workload characteristics and performance requirements.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerCPUUtilization

Dimensions: ContainerName, ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of CPU units being used by a container.

Intent: This alarm is used to detect high task CPU utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for CPU utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some containers, consistently high CPU utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical CPU utilization data to identify the usage, find what CPU utilization is acceptable for your containers, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerMemoryUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high memory utilization of containers in your Amazon ECS cluster. If memory utilization is consistently high, you might need to optimize your containers or increase the memory reservation.

Intent: This alarm is used to detect high memory utilization for containers in the Amazon ECS cluster. Consistent high memory utilization can indicate that the container is under memory pressure and might need more memory resources or optimization to maintain stability.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the container's memory reservation. You can adjust this value based on your acceptable memory utilization for the containers. For some workloads, consistently high memory utilization might be normal, while for others, it might indicate memory pressure or the need for more resources.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ContainerMemoryUtilization

Dimensions: ContainerName, ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of memory units being used by a container.

Intent: This alarm is used to detect high task memory utilization.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Typically, you can set the threshold for memory utilization to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics. For some containers, consistently high memory utilization may be normal and not indicate a problem, while for others, it may be cause of concern. Analyze historical memory utilization data to identify the usage, find what memory utilization is acceptable for your tasks, and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

TaskEBSfilesystemUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of ephemeral storage being used by a task. .

Intent: This alarm is used to detect high Amazon EBS file system usage for a task.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the Amazon EBS file system size.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskEphemeralStorageUtilization

Dimensions: ClusterName

Alarm description: This alarm helps you detect high ephemeral storage utilization of tasks in your Amazon ECS cluster. If storage utilization is consistently high, you might need to optimize your storage usage or increase the storage reservation.

Intent: This alarm is used to detect high ephemeral storage utilization for tasks in the Amazon ECS cluster. Consistent high storage utilization can indicate that the task is running out of disk space and might need more storage resources or optimization to maintain proper operation.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the task's ephemeral storage reservation. You can adjust this value based on your acceptable storage utilization for the tasks. For some workloads, consistently high storage utilization might be normal, while for others, it might indicate potential disk space issues or the need for more storage.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

TaskEphemeralStorageUtilization

Dimensions: ClusterName, ServiceName

Alarm description: This alarm helps you detect the total percentage of ephemeral storage being used by a task.

Intent: This alarm is used to detect high ephemeral storage usage for a task. Consistent high ephemeral storage utilized can indicate that the disk is full and it might lead to failure of the task.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: Set the threshold to about 80% of the ephemeral storage size.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EFS

PercentIOLimit

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that the workload stays within the I/O limit available to the file system. If the metric reaches its I/O limit consistently, consider moving the application to a file system that uses Max I/O performance as mode. For troubleshooting, check clients that are connected to the file system and applications of the clients that throttles the file system.

Intent: This alarm is used to detect how close the file system is to reach the I/O limit of the General Purpose performance mode. Consistent high I/O percentage can be an indicator of the file system cannot scale with respect to I/O requests enough and the file system can be a resource bottleneck for the applications that use the file system.

Statistic: Average

Recommended threshold: 100.0

Threshold justification: When the file system reaches its I/O limit, it may respond to read and write requests slower. Therefore, it is recommended that the metric is monitored to avoid impacting applications that use the file system. The threshold can be set around 100%. However, this value can be adjusted to a lower value based on file system characteristics.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

BurstCreditBalance

Dimensions: FileSystemId

Alarm description: This alarm helps in ensuring that there is available burst credit balance for the file system usage. When there is no available burst credit, applications access to the the file system will be limited due to low throughput. If the metric drops to 0 consistently, consider changing the throughput mode to Elastic or Provisioned throughput mode.

Intent: This alarm is used to detect low burst credit balance of the file system. Consistent low burst credit balance can be an indicator of the slowing down in throughput and increase in I/O latency.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: When the file system run out of burst credits and even if the baseline throughput rate is lower, EFS continues to provide a metered throughput of 1 MiBps to all file systems. However, it is recommended that the metric is monitored for low burst credit balance to avoid the file system acting as resource bottleneck for the applications. The threshold can be set around 0 bytes.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Amazon EKS with Container Insights

node_cpu_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high CPU utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need for replacing your worker nodes with instances that have greater CPU or a need to scale the system horizontally.

Intent: This alarm helps to monitor the CPU utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_filesystem_utilization

Dimensions: ClusterName

Alarm description: This alarm helps to detect high file system utilization in the worker nodes of the EKS cluster. If the utilization is consistently high, you might need to update your worker nodes to have larger disk volume, or you might need to scale horizontally.

Intent: This alarm helps to monitor the filesystem utilization of the worker nodes in the EKS cluster. If the utilization reaches 100%, it can lead to application failure, disk I/O bottlenecks, pod eviction, or the node to become unresponsive entirely.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: If there's sufficient disk pressure (meaning that the disk is getting full), nodes are marked as not healthy, and the pods are evicted from the node. Pods on a node with disk pressure are evicted when the available file system is lower than the eviction thresholds set on the kubelet. Set the alarm threshold so that you have enough time to react before the node is evicted from the cluster.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

node_memory_utilization

Dimensions: ClusterName

Alarm description: This alarm helps in detecting high memory utilization in worker nodes of the EKS cluster. If the utilization is consistently high, it might indicate a need to scale the number of pod replicas, or optimize your application.

Intent: This alarm helps to monitor the memory utilization of the worker nodes in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_cpu_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high CPU utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the CPU limit for the affected pod.

Intent: This alarm helps to monitor the CPU utilization of the pods belonging to a Kubernetes Service in the EKS cluster, so that you can quickly identify if a service's pod is consuming higher CPU than expected.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

pod_memory_utilization_over_pod_limit

Dimensions: ClusterName, Namespace, Service

Alarm description: This alarm helps in detecting high memory utilization in pods of the EKS cluster. If the utilization is consistently high, it might indicate a need to increase the memory limit for the affected pod.

Intent: This alarm helps to monitor the memory utilization of the pods in the EKS cluster so that the system performance doesn't degrade.

Statistic: Maximum

Recommended threshold: 80.0

Threshold justification: It is recommended to set the threshold at less than or equal to 80% to allow having enough time to debug the issue before the system starts seeing impact.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon EventBridge Scheduler

TargetErrorThrottledCount

Dimensions: None

Alarm description: This alarm helps you identify target throttling. To avoid target throttling error, consider configuring flexible time windows to spread your invocation load or increasing limits with the target service.

Intent: This alarm is used to detect target throttling errors, which can cause schedule delays.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the target throttling error is consistently greater than 0, schedule delivery is delayed. For some systems, target throttling errors for a brief period of time might be normal, while for others, it might be a cause of concern. Set this alarm's threshold, `datapointsToAlarm`, and `evaluationPeriods` accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationThrottleCount

Dimensions: None

Alarm description: This alarm helps you identify invocation throttling by Amazon EventBridge Scheduler. To avoid invocation throttling errors, consider configuring flexible time windows to spread your invocation load or increasing invocations throttle limit.

Intent: This alarm is used to detect Amazon EventBridge Scheduler invocation throttling errors, which can cause schedule delays.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the invocation throttling is consistently greater than 0, schedule delivery is delayed. For some systems, invocation throttling errors for a brief period of time might be normal, while for others, it might be a cause of concern. Set this alarm's threshold, `datapointsToAlarm`, and `evaluationPeriods` accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationDroppedCount

Dimensions: None

Alarm description: This alarm helps you identify invocations dropped by Amazon EventBridge Scheduler. Consider investigating by configuring a DLQ for the schedule.

Intent: This alarm is used to detect dropped invocations by Amazon EventBridge Scheduler. If you have configured a DLQ correctly on all of your schedules, dropped invocations will appear in the DLQ and you can skip setting up this alarm.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect dropped invocations.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

InvocationsFailedToBeSentToDeadLetterCount

Dimensions: None

Alarm description: This alarm helps you identify invocations that were failed to be sent to the configured DLQ by Amazon EventBridge Scheduler. If the metric is consistently greater than 0, modify your DLQ configuration to resolve the issue. Use `InvocationsFailedToBeSentToDeadLetterCount`_metrics to determine the issue.

Intent: This alarm is used to detect invocations failed to be sent to the configured DLQ by Amazon EventBridge Scheduler.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Set the threshold to 0 to detect any invocations that were failed to be sent to the configured DLQ. Retryable errors also show up in this metric, so `datapointsToAlarm` for this alarm has been set to 15.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Kinesis Data Streams

GetRecords.IteratorAgeMilliseconds

Dimensions: StreamName

Alarm description: This alarm can detect if iterator maximum age is too high. For real-time data processing applications, configure data retention according to tolerance of the delay. This is usually within minutes. For applications that process historic data, use this metric to monitor catchup speed. A quick solution to stop data loss is to increase the retention period while you troubleshoot the issue. You can also increase the number of workers processing records in your consumer application. The most common causes for gradual iterator age increase are insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. See link for more details.

Intent: This alarm is used to detect if data in your stream is going to expire because of being preserved too long or because record processing is too slow. It helps you avoid data loss after reaching 100% of the stream retention time.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the stream retention period and tolerance of processing delay for the records. Review

your requirements and analyze historical trends, and then set the threshold to the number of milliseconds that represents a critical processing delay. If an iterator's age passes 50% of the retention period (by default, 24 hours, configurable up to 365 days), there is a risk for data loss because of record expiration. You can monitor the metric to make sure that none of your shards ever approach this limit.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

GetRecords.Success

Dimensions: StreamName

Alarm description: This metric increments whenever your consumers successfully read data from your stream. `GetRecords` doesn't return any data when it throws an exception. The most common exception is `ProvisionedThroughputExceededException` because request rate for the stream is too high, or because available throughput is already served for the given second. Reduce the frequency or size of your requests. For more information, see Streams Limits in the Amazon Kinesis Data Streams Developer Guide, and Error Retries and Exponential Backoff in AWS.

Intent: This alarm can detect if the retrieval of records from the stream by consumers is failing. By setting an alarm on this metric, you can proactively detect any issues with data consumption, such as increased error rates or a decline in successful retrievals. This allows you to take timely actions to resolve potential problems and maintain a smooth data processing pipeline.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of retrieving records from the stream, set the threshold based on your application’s tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical GetRecords metric data as reference for the acceptable failure rate. You should also consider retries when setting the threshold because failed records can be retried. This helps to prevent transient spikes from triggering unnecessary alerts.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecord.Success

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed `PutRecord` operations breaches the threshold. Investigate the data producer logs to find the root causes of the failures. The most common reason is insufficient provisioned throughput on the shard that caused the `ProvisionedThroughputExceededException`. It happens because the request rate for the stream is too high, or the throughput attempted to be ingested into the shard is too high. Reduce the frequency or size of your requests. For more information, see Streams Limits and Error Retries and Exponential Backoff in AWS.

Intent: This alarm can detect if ingestion of records into the stream is failing. It helps you identify issues in writing data to the stream. By setting an alarm on this metric, you can proactively detect any issues of producers in publishing data to the stream, such as increased error rates or a decrease in successful records being published. This enables you to take timely actions to address potential problems and maintain a reliable data ingestion process.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the importance of data ingestion and processing to your service, set the threshold based on your application's tolerance for failed records. The threshold should be the corresponding percentage of successful operations. You can use historical PutRecord metric data as reference for the acceptable failure rate. You should also consider retries when setting the threshold because failed records can be retried.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

PutRecords.FailedRecords

Dimensions: StreamName

Alarm description: This alarm detects when the number of failed `PutRecords` exceeds the threshold. Kinesis Data Streams attempts to process all records in each `PutRecords` request, but a single record failure does not stop the processing of subsequent records. The main reason for these failures is exceeding the throughput of a stream or an individual shard. Common causes are traffic spikes and network latencies that cause records to arrive to the stream unevenly. You should detect unsuccessfully processed records and retry them in a subsequent call. Refer to Handling Failures When Using PutRecords for more details.

Intent: This alarm can detect consistent failures when using batch operation to put records to your stream. By setting an alarm on this metric, you can proactively detect an increase in failed records, enabling you to take timely actions to address the underlying problems and make sure a smooth and reliable data ingestion process.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to the number of failed records reflecting the tolerance of the the application for failed records. You can use historical data as reference for the acceptable failure value. You should also consider retries when setting the threshold because failed records can be retried in subsequent PutRecords calls.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReadProvisionedThroughputExceeded

Dimensions: StreamName

Alarm description: The alarm tracks the number of records that result in read throughput capacity throttling. If you find that you are being consistently throttled, you should consider

adding more shards to your stream to increase your provisioned read throughput. If there is more than one consumer application running on the stream, and they share the `GetRecords` limit, we recommend that you register new consumer applications through Enhanced Fan-Out. If adding more shards does not lower the number of throttles, you may have a “hot” shard that is being read from more than other shards are. Enable enhanced monitoring, find the “hot” shard, and split it.

Intent: This alarm can detect if consumers are throttled when they exceed your provisioned read throughput (determined by the number of shards you have). In that case, you won’t be able to read from the stream, and the stream can start backing up.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried and hence setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact reading from the stream and should trigger the alarm. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SubscribeToShardEvent.MillisBehindLatest

Dimensions: StreamName, ConsumerName

Alarm description: This alarm detects when the delay of record processing in the application breaches the threshold. Transient problems such as API operation failures to a downstream application can cause a sudden increase in the metric. You should investigate if they consistently happen. A common cause is the consumer is not processing records fast enough because of insufficient physical resources or record processing logic that has not scaled with an increase in stream throughput. Blocking calls in critical path is often the cause of slowdowns in record processing. You can increase your parallelism by increasing the number of shards. You should also confirm underlying processing nodes have sufficient physical resources during peak demand.

Intent: This alarm can detect delay in the subscription to shard event of the stream. This indicates a processing lag and can help identify potential issues with the consumer application's performance or the overall stream's health. When the processing lag becomes significant, you should investigate and address any bottlenecks or consumer application inefficiencies to make sure real-time data processing and minimize data backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the delay that your application can tolerate. Review your application's requirements and analyze historical trends, and then select a threshold accordingly. When the SubscribeToShard call succeeds, your consumer starts receiving SubscribeToShardEvent events over the persistent connection for up to 5 minutes, after which time you need to call SubscribeToShard again to renew the subscription if you want to continue to receive records.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

WriteProvisionedThroughputExceeded

Dimensions: StreamName

Alarm description: This alarm detects when the number of records resulting in write throughput capacity throttling reached the threshold. When your producers exceed your provisioned write throughput (determined by the number of shards you have), they are throttled and you won’t be able to put records to the stream. To address consistent throttling, you should consider adding shards to your stream. This raises your provisioned write throughput and prevents future throttling. You should also consider partition key choice when ingesting records. Random partition key is preferred because it spreads records evenly across the shards of the stream, whenever possible.

Intent: This alarm can detect if your producers are being rejected for writing records because of throttling of the stream or shard. If your stream is in Provisioned mode, then setting this alarm helps you proactively take actions when the data stream reaches its limits, allowing you

to optimize the provisioned capacity or take appropriate scaling actions to avoid data loss and maintain smooth data processing.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Usually throttled requests can be retried, so setting the threshold to zero makes the alarm too sensitive. However, consistent throttling can impact writing to the stream, and you should set the alarm threshold to detect this. Set the threshold to a percentage according to the throttled requests for the application and retry configurations.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda

ClaimedAccountConcurrency

Dimensions: None

Alarm description: This alarm helps to monitor if the concurrency of your Lambda functions is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. Request a concurrency increase in this Region.
2. Identify and reduce any unused reserved concurrency or provisioned concurrency.
3. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
4. Increase the batch size of the functions, so that more messages are processed by each function invocation.

Intent: This alarm can proactively detect if the concurrency of your Lambda functions is approaching the Region-level concurrency quota of your account, so that you can act on it. Functions are throttled if `ClaimedAccountConcurrency` reaches the Region-level

concurrency quota of the account. If you are using Reserved Concurrency (RC) or Provisioned Concurrency (PC), this alarm gives you more visibility on concurrency utilization than an alarm on `ConcurrentExecutions` would.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of about 90% of the concurrency quota set for the account in the Region, and use the result as the threshold value. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you should check the quota of your account from the Service Quotas dashboard.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Errors

Dimensions: FunctionName

Alarm description: This alarm detects high error counts. Errors includes the exceptions thrown by the code as well as exceptions thrown by the Lambda runtime. You can check the logs related to the function to diagnose the issue.

Intent: The alarm helps detect high error counts in function invocations.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value can depend on the tolerance for errors in your application. Understand the criticality of the invocations that the function is handling. For some applications, any error might be unacceptable, while other applications might allow for a certain margin of error.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: GREATER_THAN_THRESHOLD

Throttles

Dimensions: FunctionName

Alarm description: This alarm detects a high number of throttled invocation requests. Throttling occurs when there is no concurrency is available for scale up. There are several approaches to resolve this issue. 1) Request a concurrency increase from AWS Support in this Region. 2) Identify performance issues in the function to improve the speed of processing and therefore improve throughput. 3) Increase the batch size of the function, so that more messages are processed by each function invocation.

Intent: The alarm helps detect a high number of throttled invocation requests for a Lambda function. It is important to know if requests are constantly getting rejected due to throttling and if you need to improve Lambda function performance or increase concurrency capacity to avoid constant throttling.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to a number greater than zero. The exact value of the threshold can depend on the tolerance of the application. Set the threshold according to its usage and scaling requirements of the function.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

Duration

Dimensions: FunctionName

Alarm description: This alarm detects long duration times for processing an event by a Lambda function. Long durations might be because of changes in function code making the function take longer to execute, or the function's dependencies taking longer.

Intent: This alarm can detect a long running duration of a Lambda function. High runtime duration indicates that a function is taking a longer time for invocation, and can also impact the concurrency capacity of invocation if Lambda is handling a higher number of events. It is critical to know if the Lambda function is constantly taking longer execution time than expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The threshold for the duration depends on your application and workloads and your performance requirements. For high-performance requirements, set the threshold to a shorter time to see if the function is meeting expectations. You can also analyze historical data for duration metrics to see the if the time taken matches the performance expectation of the function, and then set the threshold to a longer time than the historical average. Make sure to set the threshold lower than the configured function timeout.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

ConcurrentExecutions

Dimensions: FunctionName

Alarm description: This alarm helps to monitor if the concurrency of the function is approaching the Region-level concurrency limit of your account. A function starts to be throttled if it reaches the concurrency limit. You can take the following actions to avoid throttling.

1. Request a concurrency increase in this Region.
2. Identify performance issues in the functions to improve the speed of processing and therefore improve throughput.
3. Increase the batch size of the functions, so that more messages are processed by each function invocation.

To get better visibility on reserved concurrency and provisioned concurrency utilization, set an alarm on the new metric `ClaimedAccountConcurrency` instead.

Intent: This alarm can proactively detect if the concurrency of the function is approaching the Region-level concurrency quota of your account, so that you can act on it. A function is throttled if it reaches the Region-level concurrency quota of the account.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold to about 90% of the concurrency quota set for the account in the Region. By default, your account has a concurrency quota of 1,000 across all functions in a Region. However, you can check the quota of your account, as it can be increased by contacting AWS support.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

Lambda Insights

We recommend setting best-practice alarms for the following Lambda Insights metrics.

memory_utilization

Dimensions: function_name

Alarm description: This alarm is used to detect if the memory utilization of a lambda function is approaching the configured limit. For troubleshooting, you can try to 1) Optimize your code. 2) Rightly size your memory allocation by accurately estimating the memory requirements. You can refer to Lambda Power Tuning for the same. 3) Use connection pooling. Refer to Using Amazon RDS Proxy with Lambda for the connection pooling for RDS database. 4) You can also consider designing your functions to avoid storing large amounts of data in memory between invocations.

Intent: This alarm is used to detect if the memory utilization for the Lambda function is approaching the configured limit.

Statistic: Average

Threshold Suggestion: 90.0

Threshold Justification: Set the threshold to 90% to get an alert when memory utilization exceeds 90% of the allocated memory. You can adjust this to a lower value if you have a concern for the workload for memory utilization. You can also check the historical data for this metric and set the threshold accordingly.

Period: 60

Datapoints to alarm: 10

Evaluation Periods: 10

ComparisonOperator: GREATER_THAN_THRESHOLD

Amazon VPC (`AWS/NATGateway`)

ErrorPortAllocation

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when the NAT Gateway is unable to allocate ports to new connections. To resolve this issue, see Resolve port allocation errors on NAT Gateway.

Intent: This alarm is used to detect if the NAT gateway could not allocate a source port.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the value of ErrorPortAllocation is greater than zero, that means too many concurrent connections to a single popular destination are open through NATGateway.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

PacketsDropCount

Dimensions: NatGatewayId

Alarm description: This alarm helps to detect when packets are dropped by NAT Gateway. This might happen because of an issue with NAT Gateway, so check AWS service health dashboard for the status of AWS NAT Gateway in your Region. This can help you correlate the network issue related to traffic using NAT gateway.

Intent: This alarm is used to detect if packets are being dropped by NAT Gateway.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: You should calculate the value of 0.01 percent of the total traffic on the NAT Gateway and use that result as the threshold value. Use historical data of the traffic on NAT Gateway to determine the threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

AWS Private Link (`AWS/PrivateLinkEndpoints`)

PacketsDropped

Dimensions: VPC Id, VPC Endpoint Id, Endpoint Type, Subnet Id, Service Name

Alarm description: This alarm helps to detect if the endpoint or endpoint service is unhealthy by monitoring the number of packets dropped by the endpoint. Note that packets larger than 8500 bytes that arrive at the VPC endpoint are dropped. For troubleshooting, see connectivity problems between an interface VPC endpoint and an endpoint service.

Intent: This alarm is used to detect if the endpoint or endpoint service is unhealthy.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold according to the use case. If you want to be aware of the unhealthy status of the endpoint or endpoint service, you should set the threshold low so that you get a chance to fix the issue before a huge data loss. You can use historical data to understand the tolerance for dropped packets and set the threshold accordingly.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

AWS Private Link (`AWS/PrivateLinkServices`)

RstPacketsSent

Dimensions: Service Id, Load Balancer Arn, Az

Alarm description: This alarm helps you detect unhealthy targets of an endpoint service based on the number of reset packets that are sent to endpoints. When you debug connection errors with a consumer of your service, you can validate whether the service is resetting connections with the RstPacketsSent metric, or if something else is failing on the network path.

Intent: This alarm is used to detect unhealthy targets of an endpoint service.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The threshold depends on the use case. If your use case can tolerate targets being unhealthy, you can set the threshold high. If the use case can't tolerate unhealthy targets you can set the threshold very low.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon RDS

CPUUtilization

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor consistent high CPU utilization. CPU utilization measures non-idle time. Consider using Enhanced Monitoring or Performance Insights to review which wait time is consuming the most of the CPU time (`guest`, `irq`, `wait`, `nice`, and so on) for MariaDB, MySQL, Oracle, and PostgreSQL. Then evaluate which queries consume the highest amount of CPU. If you can't tune your workload, consider moving to a larger DB instance class.

Intent: This alarm is used to detect consistent high CPU utilization in order to prevent very high response time and time-outs. If you want to check micro-bursting of CPU utilization you can set a lower alarm evaluation time.

Statistic: Average

Recommended threshold: 90.0

Threshold justification: Random spikes in CPU consumption might not hamper database performance, but sustained high CPU can hinder upcoming database requests. Depending on the overall database workload, high CPU at your RDS/Aurora instance can degrade the overall performance.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DatabaseConnections

Dimensions: DBInstanceIdentifier

Alarm description: This alarm detects a high number of connections. Review existing connections and terminate any that are in `sleep` state or that are improperly closed. Consider using connection pooling to limit the number of new connections. Alternatively, increase the DB instance size to use a class with more memory and hence a higher default value for

`max_connections` or increase the `max_connections` value in RDS and Aurora MySQL and PostgreSQL for the current class if it can support your workload.

Intent: This alarm is used to help prevent rejected connections when the maximum number of DB connections is reached. This alarm is not recommended if you frequently change DB instance class, because doing so changes the memory and default maximum number of connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The number of connections allowed depends on the size of your DB instance class and database engine-specific parameters related to processes/connections. You should calculate a value between 90-95% of the maximum number of connections for your database and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

EBSByteBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a low percentage of throughput credits remaining. For troubleshooting, check latency problems in RDS.

Intent: This alarm is used to detect a low percentage of throughput credits remaining in the burst bucket. Low byte balance percentage can cause throughput bottleneck issues. This alarm is not recommended for Aurora PostgreSQL instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: A throughput credit balance below 10% is considered to be poor and you should set the threshold accordingly. You can also set a lower threshold if your application can tolerate a lower throughput for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

EBSIOBalance%

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low percentage of IOPS credits remaining. For troubleshooting, see latency problems in RDS.

Intent: This alarm is used to detect a low percentage of I/O credits remaining in the burst bucket. Low IOPS balance percentage can cause IOPS bottleneck issues. This alarm is not recommended for Aurora instances.

Statistic: Average

Recommended threshold: 10.0

Threshold justification: An IOPS credits balance below 10% is considered to be poor and you can set the threshold accordingly. You can also set a lower threshold, if your application can tolerate a lower IOPS for the workload.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

FreeableMemory

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low freeable memory which can mean that there is a spike in database connections or that your instance may be under high memory pressure. Check for memory pressure by monitoring the CloudWatch metrics for `SwapUsage`in addition to `FreeableMemory`. If the instance memory consumption is frequently too high, this indicates that you should check your workload or upgrade your instance class. For Aurora reader

DB instance, consider adding additional reader DB instances to the cluster. For information about troubleshooting Aurora, see freeable memory issues.

Intent: This alarm is used to help prevent running out of memory which can result in rejected connections.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Depending on the workload and instance class, different values for the threshold can be appropriate. Ideally, available memory should not go below 25% of total memory for prolonged periods. For Aurora, you can set the threshold close to 5%, because the metric approaching 0 means that the DB instance has scaled up as much as it can. You can analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_THRESHOLD

FreeLocalStorage

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor low free local storage. Aurora PostgreSQL-Compatible Edition uses local storage for storing error logs and temporary files. Aurora MySQL uses local storage for storing error logs, general logs, slow query logs, audit logs, and non-InnoDB temporary tables. These local storage volumes are backed by Amazon EBS Store and can be extended by using a larger DB instance class. For troubleshooting, check Aurora PostgreSQL-Compatible and MySQL-Compatible.

Intent: This alarm is used to detect how close the Aurora DB instance is to reaching the local storage limit, if you do not use Aurora Serverless v2 or higher. Local storage can reach capacity when you store non-persistent data, such as temporary table and log files, in the local storage. This alarm can prevent an out-of-space error that occurs when your DB instance runs out of local storage.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 10%-20% of the amount of storage available based on velocity and trend of volume usage, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

FreeStorageSpace

Dimensions: DBInstanceIdentifier

Alarm description: This alarm watches for a low amount of available storage space. Consider scaling up your database storage if you frequently approach storage capacity limits. Include some buffer to accommodate unforeseen increases in demand from your applications. Alternatively, consider enabling RDS storage auto scaling. Additionally, consider freeing up more space by deleting unused or outdated data and logs. For further information, check RDS run out of storage document and PostgreSQL storage issues document.

Intent: This alarm helps prevent storage full issues. This can prevent downtime that occurs when your database instance runs out of storage. We do not recommend using this alarm if you have storage auto scaling enabled, or if you frequently change the storage capacity of the database instance.

Statistic: Minimum

Recommended threshold: Depends on your situation

Threshold justification: The threshold value will depend on the currently allocated storage space. Typically, you should calculate the value of 10 percent of the allocated storage space and use that result as the threshold value.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

MaximumUsedTransactionIDs

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps prevent transaction ID wraparound for PostgreSQL. Refer to the troubleshooting steps in this blog to investigate and resolve the issue. You can also refer to this blog to familiarize yourself further with autovacuum concepts, common issues and best practices.

Intent: This alarm is used to help prevent transaction ID wraparound for PostgreSQL.

Statistic: Average

Recommended threshold: 1.0E9

Threshold justification: Setting this threshold to 1 billion should give you time to investigate the problem. The default autovacuum_freeze_max_age value is 200 million. If the age of the oldest transaction is 1 billion, autovacuum is having a problem keeping this threshold below the target of 200 million transaction IDs.

Period: 60

Datapoints to alarm: 1

Evaluation periods: 1

Comparison Operator: GREATER_THAN_THRESHOLD

ReadLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high read latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck. For Aurora, you can switch to an instance class that has I/O-Optimized storage configuration. See Planning I/O in Aurora for guidance.

Intent: This alarm is used to detect high read latency. Database disks normally have a low read/write latency, but they can have issues that can cause high latency operations.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Read latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have higher latency for read operations. Review the criticality and requirements of read latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

ReplicaLag

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you understand the number of seconds a replica is behind the primary instance. A PostgreSQL Read Replica reports a replication lag of up to five minutes if there are no user transactions occurring on the source database instance. When the ReplicaLag metric reaches 0, the replica has caught up to the primary DB instance. If the ReplicaLag metric returns -1, then replication is currently not active. For guidance related to RDS PostgreSQL, see replication best practices and for troubleshooting `ReplicaLag` and related errors, see troubleshooting ReplicaLag.

Intent: This alarm can detect the replica lag which reflects the data loss that could happen in case of a failure of the primary instance. If the replica gets too far behind the primary and the primary fails, the replica will be missing data that was in the primary instance.

Statistic: Maximum

Recommended threshold: 60.0

Threshold justification: Typically, the acceptable lag depends on the application. We recommend no more than 60 seconds.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: GREATER_THAN_THRESHOLD

WriteLatency

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high write latency. If storage latency is high, it's because the workload is exceeding resource limits. You can review I/O utilization relative to instance and allocated storage configuration. Refer to troubleshoot the latency of Amazon EBS volumes caused by an IOPS bottleneck. For Aurora, you can switch to an instance class that has I/O-Optimized storage configuration. See Planning I/O in Aurora for guidance.

Intent: This alarm is used to detect high write latency. Although database disks typically have low read/write latency, they may experience problems that cause high latency operations. Monitoring this will assure you the disk latency is as low as expected.

Statistic: p90

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on your use case. Write latencies higher than 20 milliseconds are likely a cause for investigation. You can also set a higher threshold if your application can have a higher latency for write operations. Review the criticality and requirements of write latency and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

DBLoad

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor high DB load. If the number of processes exceed the number of vCPUs, the processes start queuing. When the queuing increases, the performance is impacted. If the DB load is often above the maximum vCPU, and the primary

wait state is CPU, the CPU is overloaded. In this case, you can monitor `CPUUtilization`, `DBLoadCPU` and queued tasks in Performance Insights/Enhanced Monitoring. You might want to throttle connections to the instance, tune any SQL queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve.

Intent: This alarm is used to detect a high DB load. High DB load can cause performance issues in the DB instance. This alarm is not applicable to serverless DB instances.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The maximum vCPU value is determined by the number of vCPU (virtual CPU) cores for your DB instance. Depending on the maximum vCPU, different values for the threshold can be appropriate. Ideally, DB load should not go above vCPU line.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

AuroraVolumeBytesLeftTotal

Dimensions: DBClusterIdentifier

Alarm description: This alarm helps to monitor low remaining total volume. When the total volume left reaches the size limit, the cluster reports an out-of-space error. Aurora storage automatically scales with the data in the cluster volume and expands up to 128 TiB or 64 TiB depending on the DB engine version. Consider reducing storage by dropping tables and databases that you no longer need. For more information, check storage scaling.

Intent: This alarm is used to detect how close the Aurora cluster is to the volume size limit. This alarm can prevent an out-of-space error that occurs when your cluster runs out of space. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 10%-20% of the actual size limit based on velocity and trend of volume usage increase, and then use that result as the threshold value to proactively take action before the volume reaches its limit.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

AuroraBinlogReplicaLag

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor the error state of Aurora writer instance replication. For more information, see Replicating Aurora MySQL DB clusters across AWS Regions. For troubleshooting, see Aurora MySQL replication issues.

Intent: This alarm is used to detect whether the writer instance is in an error state and can't replicate the source. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: -1.0

Threshold justification: We recommend that you use -1 as the threshold value because Aurora MySQL publishes this value if the replica is in an error state.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

BlockedTransactions

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a high blocked transaction count in an Aurora DB instance. Blocked transactions can end in either a rollback or a commit. High concurrency,

idles in transaction, or long running transactions can lead to blocked transactions. For troubleshooting, see Aurora MySQL documentation.

Intent: This alarm is used to detect a high count of blocked transactions in an Aurora DB instance in order to prevent transaction rollbacks and performance degradation.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate 5% of all transactions of your instance using the `ActiveTransactions` metric and use that result as the threshold value. You can also review the criticality and requirements of blocked transactions and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

BufferCacheHitRatio

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps you monitor a consistent low cache hit ratio of the Aurora cluster. A low hit ratio indicates that your queries on this DB instance are frequently going to disk. For troubleshooting, investigate your workload to see which queries are causing this behavior, and see the DB instance RAM recommendations document.

Intent: This alarm is used to detect consistent low cache hit ratio in order to prevent a sustained performance decrease in the Aurora instance.

Statistic: Average

Recommended threshold: 80.0

Threshold justification: You can set the threshold for buffer cache hit ratio to 80%. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 10

Evaluation periods: 10

Comparison Operator: LESS_THAN_THRESHOLD

EngineUptime

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor low downtime of the writer DB instance. The writer DB instance can go down due to a reboot, maintenance, upgrade, or failover. When the uptime reaches 0 because of a failover in the cluster, and the cluster has one or more Aurora Replicas, then an Aurora Replica is promoted to the primary writer instance during a failure event. To increase the availability of your DB cluster, consider creating one or more Aurora Replicas in two or more different Availability Zones. For more information check factors that influence Aurora downtime.

Intent: This alarm is used to detect whether the Aurora writer DB instance is in downtime. This can prevent long-running failure in the writer instance that occurs because of a crash or failover.

Statistic: Average

Recommended threshold: 0.0

Threshold justification: A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 60 seconds, and often less than 30 seconds.

Period: 60

Datapoints to alarm: 2

Evaluation periods: 2

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

RollbackSegmentHistoryListLength

Dimensions: DBInstanceIdentifier

Alarm description: This alarm helps to monitor a consistent high rollback segment history length of an Aurora instance. A high InnoDB history list length indicates that a large number of

old row versions, queries and database shutdowns have become slower. For more information and troubleshooting, see the InnoDB history list length increased significantly documentation.

Intent: This alarm is used to detect consistent high rollback segment history length. This can help you prevent sustained performance degradation and high CPU usage in the Aurora instance. This alarm is recommended only for Aurora MySQL.

Statistic: Average

Recommended threshold: 1000000.0

Threshold justification: Setting this threshold to 1 million should give you time to investigate the problem. However, you can adjust this value based on your acceptable performance level and workload characteristics.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

StorageNetworkThroughput

Dimensions: DBClusterIdentifier, Role=WRITER

Alarm description: This alarm helps to monitor high storage network throughput. If storage network throughput passes the total network bandwidth of the EC2 instance, it can lead to high read and write latency, which can cause degraded performance. You can check your EC2 instance type from AWS Console. For troubleshooting, check any changes on write/read latencies and evaluate if you've also hit an alarm on this metric. If that is the case, evaluate your workload pattern during the times that the alarm was triggered. This can help you identify if you can optimize y our workload to reduce the total amount of network traffic. If this is not possible, you might need to consider scaling your instance.

Intent: This alarm is used to detect high storage network throughput. Detecting high throughput can prevent network packet drops and degraded performance.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: You should calculate about 80%-90% of the total network bandwidth of the EC2 instance type, and then use that result as the threshold value to proactively take action before the network packets are affected. You can also review the criticality and requirements of storage network throughput and analyze the historical behavior of this metric to determine sensible threshold levels.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon Route 53 Public Data Plane

HealthCheckStatus

Dimensions: HealthCheckId

Alarm description: This alarm helps to detect unhealthy endpoints as per health checkers. To understand the reason for a failure that results in unhealthy status, use the Health Checkers tab in the Route 53 Health Check Console to view the status from each Region as well as the last failure of the health check. The status tab also displays the reason that the endpoint is reported as unhealthy. Refer to troubleshooting steps.

Intent: This alarm uses Route53 health checkers to detect unhealthy endpoints.

Statistic: Average

Recommended threshold: 1.0

Threshold justification: The status of the endpoint is reported as 1 when it's healthy. Everything less than 1 is unhealthy.

Period: 60

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Amazon S3

4xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps us report the total number of 4xx error status codes that are made in response to client requests. 403 error codes might indicate an incorrect IAM policy, and 404 error codes might indicate mis-behaving client application, for example. Enabling S3 server access logging on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code. To understand more about the error code, see Error Responses.

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: The recommended threshold is to detect if more than 5% of total requests are getting 4XX errors. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: BucketName, FilterId

Alarm description: This alarm helps you detect a high number of server-side errors. These errors indicate that a client made a request that the server couldn't complete. This can help you correlate the issue your application is facing because of S3. For more information to help you

efficiently handle or reduce errors, see Optimizing performance design patterns. Errors might also be caused by an the issue with S3, check AWS service health dashboard for the status of Amazon S3 in your Region.

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XXError. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

OperationsFailedReplication

Dimensions: SourceBucket, DestinationBucket, RuleId

Alarm description: This alarm helps in understanding a replication failure. This metric tracks the status of new objects replicated using S3 CRR or S3 SRR, and also tracks existing objects replicated using S3 batch replication. See Replication troubleshooting for more details.

Intent: This alarm is used to detect if there is a failed replication operation.

Statistic: Maximum

Recommended threshold: 0.0

Threshold justification: This metric emits a value of 0 for successful operations, and nothing when there are no replication operations carried out for the minute. When the metric emits a value greater than 0, the replication operation is unsuccessful.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

S3ObjectLambda

4xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps us report the total number of 4xx error status code that are made in response to client requests. Enabling S3 server access logging on a temporary basis will help you to pinpoint the issue's origin using the fields HTTP status and Error Code.

Intent: This alarm is used to create a baseline for typical 4xx error rates so that you can look into any abnormalities that might indicate a setup issue.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

5xxErrors

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps to detect high number of server-side errors. These errors indicate that a client made a request that the server couldn't complete. These errors might be caused by an issue with S3, check AWS service health dashboard for the status of Amazon S3 in your Region. This can help you correlate the issue your application is facing because of S3. For information to help you efficiently handle or reduce these errors, see Optimizing performance design patterns.

Intent: This alarm can help to detect if the application is experiencing issues due to 5xx errors.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 5XX errors. However, you can tune the threshold to suit the traffic of the requests, as well as acceptable error rates. You can also analyze historical data to see what is the acceptable error rate for the application workload, and tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

LambdaResponse4xx

Dimensions: AccessPointName, DataSourceARN

Alarm description: This alarm helps you detect and diagnose failures (500s) in calls to S3 Object Lambda. These errors can be caused by errors or misconfigurations in the Lambda function responsible for responding to your requests. Investigating the CloudWatch Log Streams of the Lambda function associated with the Object Lambda Access Point can help you pinpoint the issue's origin based on the response from S3 Object Lambda.

Intent: This alarm is used to detect 4xx client errors for WriteGetObjectResponse calls.

Statistic: Average

Recommended threshold: 0.05

Threshold justification: We recommend setting the threshold to detect if more than 5% of total requests are getting 4XXError. Frequently occurring 4XX errors should be alarmed. However, setting a very low value for the threshold can cause alarm to be too sensitive. You can also tune the threshold to suit to the load of the requests, accounting for an acceptable level of 4XX errors. You can also analyze historical data to find the acceptable error rate for the application workload, and then tune the threshold accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SNS

NumberOfMessagesPublished

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages published is too low. For troubleshooting, check why the publishers are sending less traffic.

Intent: This alarm helps you proactively monitor and detect significant drops in notification publishing. This helps you identify potential issues with your application or business processes, so that you can take appropriate actions to maintain the expected flow of notifications. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages published should be in line with the expected number of published messages for your application. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsDelivered

Dimensions: TopicName

Alarm description: This alarm can detect when the number of SNS messages delivered is too low. This could be because of unintentional unsubscribing of an endpoint, or because of an SNS event that causes messages to experience delay.

Intent: This alarm helps you detect a drop in the volume of messages delivered. You should create this alarm if you expect your system to have a minimum traffic that it is serving.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The number of messages delivered should be in line with the expected number of messages produced and the number of consumers. You can also analyze the historical data, trends and traffic to find the right threshold.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: LESS_THAN_THRESHOLD

NumberOfNotificationsFailed

Dimensions: TopicName

Alarm description: This alarm can detect when the number of failed SNS messages is too high. To troubleshoot failed notifications, enable logging to CloudWatch Logs. Checking the logs can help you find which subscribers are failing, as well as the status codes they are returning.

Intent: This alarm helps you proactively find issues with the delivery of notifications and take appropriate actions to address them.

Statistic: Sum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the impact of failed notifications. Review the SLAs provided to your end users, fault tolerance and criticality of notifications and analyze historical data, and then select a threshold accordingly. The number of notifications failed should be 0 for topics that have only SQS, Lambda or Firehose subscriptions.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidAttributes

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid attributes or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid attributes are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid attributes are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFilteredOut-InvalidMessageBody

Dimensions: TopicName

Alarm description: This alarm helps to monitor and resolve potential problems with the publisher or subscribers. Check if a publisher is publishing messages with invalid message bodies, or if an inappropriate filter is applied to a subscriber. You can also analyze CloudWatch Logs to help find the root cause of the issue.

Intent: The alarm is used to detect if the published messages are not valid or if inappropriate filters have been applied to a subscriber.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: Invalid message bodies are almost always a mistake by the publisher. We recommend to set the threshold to 0 because invalid message bodies are not expected in a healthy system.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsRedrivenToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor the number of messages that are moved to a dead-letter queue.

Intent: The alarm is used to detect messages that moved to a dead-letter queue. We recommend that you create this alarm when SNS is coupled with SQS, Lambda or Firehose.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: In a healthy system of any subscriber type, messages should not be moved to the dead-letter queue. We recommend that you be notified if any messages land in the queue, so that you can identify and address the root cause, and potentially redrive the messages in the dead-letter queue to prevent data loss.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

NumberOfNotificationsFailedToRedriveToDlq

Dimensions: TopicName

Alarm description: This alarm helps to monitor messages that couldn't be moved to a dead-letter queue. Check whether your dead-letter queue exists and that it's configured correctly. Also, verify that SNS has permissions to access the dead-letter queue. Refer to the dead-letter queue documentation to learn more.

Intent: The alarm is used to detect messages that couldn't be moved to a dead-letter queue.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: It's almost always a mistake if messages can't be moved to the dead-letter queue. The recommendation for the threshold is 0, meaning all messages that fail processing must be able to reach the dead-letter queue when the queue has been configured.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSMonthToDateSpentUSD

Dimensions: TopicName

Alarm description: The alarm helps to monitor if you have a sufficient quota in your account for SNS to be able to deliver messages. If you reach your quota, SNS won't be able to deliver SMS messages. For information about setting your monthly SMS spend quota, or for information about requesting a spend quota increase with AWS, see Setting SMS messaging preferences.

Intent: This alarm is used to detect if you have a sufficient quota in your account for your SMS messages to be delivered successfully.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold in accordance with the quota (Account spend limit) for the account. Choose a threshold which informs you early enough that you are reaching your quota limit so that you have time to request an increase.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

SMSSuccessRate

Dimensions: TopicName

Alarm description: This alarm helps to monitor the rate of failing SMS message deliveries. You can set up Cloudwatch Logs to understand the nature of the failure and take action based on that.

Intent: This alarm is used to detect failing SMS message deliveries.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: Set the threshold for the alarm in line with your tolerance for failing SMS message deliveries.

Period: 60

Datapoints to alarm: 5

Evaluation periods: 5

Comparison Operator: GREATER_THAN_THRESHOLD

Amazon SQS

ApproximateAgeOfOldestMessage

Dimensions: QueueName

Alarm description: This alarm watches the age of the oldest message in the queue. You can use this alarm to monitor if your consumers are processing SQS messages at the desired speed. Consider increasing the consumer count or consumer throughput to reduce message age. This metric can be used in combination with `ApproximateNumberOfMessagesVisible` to determine how big the queue backlog is and how quickly messages are being processed. To prevent messages from being deleted before processed, consider configuring the dead-letter queue to sideline potential poison pill messages.

Intent: This alarm is used to detect whether the age of the oldest message in the QueueName queue is too high. High age can be an indication that messages are not processed quickly enough or that there are some poison-pill messages that are stuck in the queue and can't be processed.

Statistic: Maximum

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected message processing time. You can use historical data to calculate the average message processing time, and then set the threshold to 50% higher than the maximum expected SQS message processing time by queue consumers.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesNotVisible

Dimensions: QueueName

Alarm description: This alarm helps to detect a high number of in-flight messages with respect to `QueueName`. For troubleshooting, check message backlog decreasing.

Intent: This alarm is used to detect a high number of in-flight messages in the queue. If consumers do not delete messages within the visibility timeout period, when the queue is polled, messages reappear in the queue. For FIFO queues, there can be a maximum of 20,000 in-flight messages. If you reach this quota, SQS returns no error messages. A FIFO queue looks through the first 20k messages to determine available message groups. This means that if you have a backlog of messages in a single message group, you cannot consume messages from other message groups that were sent to the queue at a later time until you successfully consume the messages from the backlog.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: The recommended threshold value for this alarm is highly dependent on the expected number of messages in flight. You can use historical data to calculate the maximum expected number of messages in flight and set the threshold to 50% over this value. If consumers of the queue are processing but not deleting messages from the queue, this number will suddenly increase.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

ApproximateNumberOfMessagesVisible

Dimensions: QueueName

Alarm description: This alarm watches for the message queue backlog to be bigger than expected, indicating that consumers are too slow or there are not enough consumers. Consider increasing the consumer count or speeding up consumers, if this alarm goes into ALARM state.

Intent: This alarm is used to detect whether the message count of the active queue is too high and consumers are slow to process the messages or there are not enough consumers to process them.

Statistic: Average

Recommended threshold: Depends on your situation

Threshold justification: An unexpectedly high number of messages visible indicates that messages are not being processed by a consumer at the expected rate. You should consider historical data when you set this threshold.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: GREATER_THAN_OR_EQUAL_TO_THRESHOLD

NumberOfMessagesSent

Dimensions: QueueName

Alarm description: This alarm helps to detect if there are no messages being sent from a producer with respect to `QueueName`. For troubleshooting, check the reason that the producer is not sending messages.

Intent: This alarm is used to detect when a producer stops sending messages.

Statistic: Sum

Recommended threshold: 0.0

Threshold justification: If the number of messages sent is 0, the producer is not sending any messages. If this queue has a low TPS, increase the number of EvaluationPeriods accordingly.

Period: 60

Datapoints to alarm: 15

Evaluation periods: 15

Comparison Operator: LESS_THAN_OR_EQUAL_TO_THRESHOLD

Site-to-Site VPN

TunnelState

Dimensions: VpnId

Alarm description: This alarm helps you understand if the state of one or more tunnels is DOWN. For troubleshooting, see VPN tunnel troubleshooting.

Intent: This alarm is used to detect if at least one tunnel is in the DOWN state for this VPN, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that at least one tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

TunnelState

Dimensions: TunnelIpAddress

Alarm description: This alarm helps you understand if the state of this tunnel is DOWN. For troubleshooting, see VPN tunnel troubleshooting.

Intent: This alarm is used to detect if the tunnel is in the DOWN state, so that you can troubleshoot the impacted VPN. This alarm will always be in the ALARM state for networks that only have a single tunnel configured.

Statistic: Minimum

Recommended threshold: 1.0

Threshold justification: A value less than 1 indicates that the tunnel is in DOWN state.

Period: 300

Datapoints to alarm: 3

Evaluation periods: 3

Comparison Operator: LESS_THAN_THRESHOLD

Alarm use cases and examples

The following sections provides examples and tutorials for alarms for common use cases.

Topics

- Create a billing alarm to monitor your estimated AWS charges
- Create a CPU usage alarm
- Create a load balancer latency alarm that sends email
- Create a storage throughput alarm that sends email
- Create an alarm on Performance Insights counter metrics from an AWS database
- Create a Log Alarm on error patterns
- Create a Log Alarm on high latency per endpoint
- Create an alarm that triggers an Auto Scaling action

Create a billing alarm to monitor your estimated AWS charges

You can monitor your estimated AWS charges by using Amazon CloudWatch. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to CloudWatch as metric data.

Billing metric data is stored in the US East (N. Virginia) Region and represents worldwide charges. This data includes the estimated charges for every service in AWS that you use, in addition to the estimated overall total of your AWS charges.

The alarm triggers when your account billing exceeds the threshold you specify. It triggers only when the current billing exceeds the threshold. It doesn't use projections based on your usage so far in the month.

If you create a billing alarm at a time when your charges have already exceeded the threshold, the alarm goes to the ALARM state immediately.

> **Note**
>
> For information about analyzing CloudWatch charges that you have already been billed for, see Analyzing, optimizing, and reducing CloudWatch costs.

Tasks

- Enabling billing alerts
- Create a billing alarm
- Deleting a billing alarm

Enabling billing alerts

Before you can create an alarm for your estimated charges, you must enable billing alerts, so that you can monitor your estimated AWS charges and create an alarm using billing metric data. After you enable billing alerts, you can't disable data collection, but you can delete any billing alarms that you created.

After you enable billing alerts for the first time, it takes about 15 minutes before you can view billing data and set billing alarms.

Requirements

- You must be signed in using account root user credentials or as an IAM user that has been given permission to view billing information.
- For consolidated billing accounts, billing data for each linked account can be found by logging in as the paying account. You can view billing data for total estimated charges and estimated charges by service for each linked account, in addition to the consolidated account.
- In a consolidated billing account, member linked account metrics are captured only if the payer account enables the **Receive Billing Alerts** preference. If you change which account is your management/payer account, you must enable the billing alerts in the new management/payer account.
- The account must not be part of the Amazon Partner Network (APN) because billing metrics are not published to CloudWatch for APN accounts. For more information, see AWS Partner Network.

To enable the monitoring of estimated charges

1. Open the AWS Billing and Cost Management console at https://console.aws.amazon.com/costmanagement/.
2. In the navigation pane, choose **Billing Preferences**.
3. By **Alert preferences** choose **Edit**.
4. Choose **Receive CloudWatch Billing Alerts**.
5. Choose **Save preferences**.

Create a billing alarm

> **Important**
>
> Before you create a billing alarm, you must set your Region to US East (N. Virginia). Billing metric data is stored in this Region and represents worldwide charges. You also must enable billing alerts for your account or in the management/payer account (if you are using consolidated billing). For more information, see Enabling billing alerts.

In this procedure, you create an alarm that sends a notification when your estimated charges for AWS exceed a defined threshold.

To create a billing alarm using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**. In **AWS Namespaces**, choose **Billing**, and then choose **Total Estimated Charge**.

   > **Note**
   >
   > If you don't see the **Billing/Total Estimated Charge** metric, enable billing alerts, and change your Region to US East (N. Virginia). For more information, see Enabling billing alerts.

5. Select the box for the **EstimatedCharges** metric, and then choose **Select metric**.

6. For **Statistic**, choose **Maximum**.
7. For **Period**, choose **6 hours**.
8. For **Threshold type**, choose **Static**.
9. For **Whenever EstimatedCharges is . . .**, choose **Greater**.
10. For **than . . .**, define the value that you want to cause your alarm to trigger. For example, **`200`** USD.

    The **EstimatedCharges** metric values are only in US dollars (USD), and the currency conversion is provided by Amazon Services LLC. For more information, see What is AWS Billing?.

    > **Note**
    >
    > After you define a threshold value, the preview graph displays your estimated charges for the current month.

11. Choose **Additional Configuration** and do the following:

    - For **Datapoints to alarm**, specify **1 out of 1**.
    - For **Missing data treatment**, choose **Treat missing data as missing**.

12. Choose **Next**.
13. Under **Notification**, make sure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the `ALARM` state. The Amazon SNS topic can include your email address so that you receive email when the billing amount crosses the threshold that you specified.

    You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

14. Choose **Next**.
15. Under **Name and description**, enter a name for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters.

    - (Optional) Enter a description of your alarm. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

16. Choose **Next**.

17. Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

Deleting a billing alarm

You can delete your billing alarm when you no longer need it.

To delete a billing alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region to US East (N. Virginia). Billing metric data is stored in this Region and reflects worldwide charges.
3. In the navigation pane, choose **Alarms**, **All alarms**.
4. Select the check box next to the alarm and choose **Actions**, **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Create a CPU usage alarm

You can create an CloudWatch alarm that sends a notification using Amazon SNS when the alarm changes state from `OK` to `ALARM`.

The alarm changes to the `ALARM` state when the average CPU use of an EC2 instance exceeds a specified threshold for consecutive specified periods.

Setting up a CPU usage alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.
5. In the **Browse** tab, choose **EC2 metrics**.
6. Choose a metric category (for example, **Per-Instance Metrics**).

7. Find the row with the instance that you want listed in the **InstanceId** column and **CPUUtilization** in the **Metric Name** column. Select the check box next to this row, and choose **Select metric**.
8. Under **Specify metric and conditions**, for **Statistic** choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95.45**).
9. Choose a period (for example, **5 minutes**).
10. Under **Conditions**, specify the following:

    a. For **Threshold type**, choose **Static**.

    b. For **Whenever CPUUtilization is**, specify **Greater**. Under **than...**, specify the threshold that is to trigger the alarm to go to ALARM state if the CPU utilization exceeds this percentage. For example, 70.

    c. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

    To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.

    d. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

    e. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Use it to choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information, see Percentile-based alarms and low data samples.
11. Choose **Next**.
12. Under **Notification**, choose **In alarm** and select an SNS topic to notify when the alarm is in ALARM state

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.
13. When finished, choose **Next**.

14. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

15. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Setting up a CPU usage alarm using the AWS CLI

Use these steps to use the AWS CLI to create a CPU usage alarm.

To create an alarm based on CPU usage

1. Set up an SNS topic. For more information, see Setting up Amazon SNS notifications.
2. Create an alarm using the put-metric-alarm command as follows.

```
aws cloudwatch put-metric-alarm --alarm-name cpu-mon --alarm-description "Alarm
 when CPU exceeds 70%" --metric-name CPUUtilization --namespace AWS/EC2 --statistic
 Average --period 300 --threshold 70 --comparison-operator GreaterThanThreshold --
dimensions  Name=InstanceId,Value=i-12345678 --evaluation-periods 2 --alarm-actions
 arn:aws:sns:us-east-1:111122223333:my-topic --unit Percent
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

    a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

    ```
    aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
     "initializing" --state-value OK
    ```

    b. Change the alarm state from `OK` to `ALARM`.

    ```
    aws cloudwatch set-alarm-state --alarm-name cpu-mon --state-reason
     "initializing" --state-value ALARM
    ```

    c. Check that you have received a notification about the alarm.

Create a load balancer latency alarm that sends email

You can set up an Amazon SNS notification and configure an alarm that monitors latency exceeding 100 ms for your Classic Load Balancer.

Setting up a latency alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Under **CloudWatch Metrics by Category**, choose the **ELB Metrics** category.
5. Select the row with the Classic Load Balancer and the **Latency** metric.
6. For the statistic, choose **Average**, choose one of the predefined percentiles, or specify a custom percentile (for example, **p95.45**).
7. For the period, choose **1 Minute**.
8. Choose **Next**.
9. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **myHighCpuAlarm**) and a description of the alarm (for example, **Alarm when Latency exceeds 100s**). Alarm names must contain only UTF-8 characters, and can't contain ASCII control characters

   The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
10. Under **Whenever**, for **is**, choose **>** and enter **0.1**. For **for**, enter **3**.
11. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.

    For **Percentiles with low samples**, choose **ignore (maintain the alarm state)** so that the alarm evaluates only situations with adequate numbers of data samples.
12. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create a new one.

To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **myHighCpuAlarm**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent.

13. Choose **Create Alarm**.

Setting up a latency alarm using the AWS CLI

Use these steps to use the AWS CLI to create a load balancer latency alarm.

To create a load balancer latency alarm

1. Set up an SNS topic. For more information, see Setting up Amazon SNS notifications.
2. Create the alarm using the put-metric-alarm command as follows:

```
aws cloudwatch put-metric-alarm --alarm-name lb-mon --alarm-description "Alarm
 when Latency exceeds 100s" --metric-name Latency --namespace AWS/ELB --statistic
 Average --period 60 --threshold 100 --comparison-operator GreaterThanThreshold --
dimensions Name=LoadBalancerName,Value=my-server --evaluation-periods 3 --alarm-
actions arn:aws:sns:us-east-1:111122223333:my-topic --unit Seconds
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

   a. Change the alarm state from INSUFFICIENT_DATA to OK.

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value OK
   ```

   b. Change the alarm state from OK to ALARM.

   ```
   aws cloudwatch set-alarm-state --alarm-name lb-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. Check that you have received an email notification about the alarm.

Create a storage throughput alarm that sends email

You can set up an SNS notification and configure an alarm that is triggered when Amazon EBS exceeds 100 MB throughput.

Setting up a storage throughput alarm using the AWS Management Console

Use these steps to use the AWS Management Console to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All Alarms**.
3. Choose **Create alarm**.
4. Under **EBS Metrics**, choose a metric category.
5. Select the row with the volume and the **VolumeWriteBytes** metric.
6. For the statistic, choose **Average**. For the period, choose **5 Minutes**. Choose **Next**.
7. Under **Alarm Threshold**, enter a unique name for the alarm (for example, **`myHighWriteAlarm`**) and a description of the alarm (for example, **`VolumeWriteBytes exceeds 100,000 KiB/s`**). The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.
8. Under **Whenever**, for **is**, choose **>** and enter **`100000`**. For **for**, enter **15** consecutive periods.

   A graphical representation of the threshold is shown under **Alarm Preview**.
9. Under **Additional settings**, for **Treat missing data as**, choose **ignore (maintain alarm state)** so that missing data points don't trigger alarm state changes.
10. Under **Actions**, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic or create one.

    To create an SNS topic, choose **New list**. For **Send notification to**, enter a name for the SNS topic (for example, **`myHighCpuAlarm`**), and for **Email list**, enter a comma-separated list of email addresses to be notified when the alarm changes to the ALARM state. Each email address is sent a topic subscription confirmation email. You must confirm the subscription before notifications can be sent to an email address.

11. Choose **Create Alarm**.

Setting up a storage throughput alarm using the AWS CLI

Use these steps to use the AWS CLI to create an alarm based on Amazon EBS throughput.

To create a storage throughput alarm

1. Create an SNS topic. For more information, see Setting up Amazon SNS notifications.
2. Create the alarm.

```
aws cloudwatch put-metric-alarm --alarm-name ebs-mon --alarm-description "Alarm
 when EBS volume exceeds 100MB throughput" --metric-name VolumeReadBytes --
namespace AWS/EBS --statistic Average --period 300 --threshold 100000000 --
comparison-operator GreaterThanThreshold --dimensions Name=VolumeId,Value=my-
volume-id --evaluation-periods 3 --alarm-actions arn:aws:sns:us-
east-1:111122223333:my-alarm-topic --insufficient-data-actions arn:aws:sns:us-
east-1:111122223333:my-insufficient-data-topic
```

3. Test the alarm by forcing an alarm state change using the set-alarm-state command.

   a. Change the alarm state from `INSUFFICIENT_DATA` to `OK`.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value OK
   ```

   b. Change the alarm state from `OK` to `ALARM`.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value ALARM
   ```

   c. Change the alarm state from `ALARM` to `INSUFFICIENT_DATA`.

   ```
   aws cloudwatch set-alarm-state --alarm-name ebs-mon --state-reason
    "initializing" --state-value INSUFFICIENT_DATA
   ```

   d. Check that you have received an email notification about the alarm.

Create an alarm on Performance Insights counter metrics from an AWS database

CloudWatch includes a **DB_PERF_INSIGHTS** metric math function which you can use to bring Performance Insights counter metrics into CloudWatch from Amazon Relational Database Service and Amazon DocumentDB (with MongoDB compatibility). **DB_PERF_INSIGHTS** also brings in the DBLoad metric at sub-minute intervals. You can set CloudWatch alarms on these metrics.

For more information about Amazon RDS Performance Insights, see Monitoring DB load with Performance Insights on Amazon RDS.

For more information about Amazon DocumentDB Performance Insights, see Monitoring with Performance Insights.

Anomaly detection is not supported for alarms based on the **DB_PERF_INSIGHTS** function.

> **Note**
>
> High-resolution metrics with sub-minute granularity retrieved by **DB_PERF_INSIGHTS** are only applicable to the **DBLoad** metric, or for operating system metrics if you have enabled Enhanced Monitoring at a higher resolution. For more information about Amazon RDS enhanced monitoring, see Monitoring OS metrics with Enhanced Monitoring..
> You can create a high-resolution alarm using the **DB_PERF_INSIGHTS** function. The maximum evaluation range for a high-resolution alarm is three hours. You can use the CloudWatch console to graph metrics retrieved with the **DB_PERF_INSIGHTS** function for any time range.

To create an alarm that's based on Performance Insights metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and then choose **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select Metric**.
5. Choose the **Add math** dropdown, and then select **All functions**, **DB_PERF_INSIGHTS** from the list.

After you choose **DB_PERF_INSIGHTS**, a math expression box appears where you apply or edit math expressions.

6. In the math expression box, enter your **DB_PERF_INSIGHTS** math expression, and then choose **Apply**.

   For example, **`DB_PERF_INSIGHTS('RDS', 'db-ABCDEFGHIJKLMNOPQRSTUVWXY1', 'os.cpuUtilization.user.avg')`**

   > **Important**
   >
   > When you use the **DB_PERF_INSIGHTS** math expression, you must specify the Unique Database Resource ID of the database. This is different than the database identifier. To find the database resource ID in the Amazon RDS console, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is displayed in the **Configuration** section.

   For information about the **DB_PERF_INSIGHTS** function and other functions that are available for metric math, see Metric math syntax and functions.

7. Choose **Select metric**.

   The **Specify metric and conditions** page appears, showing a graph and other information about the math expression that you have selected.

8. For **Whenever *`expression`* is**, specify whether the expression must be greater than, less than, or equal to the threshold. Under **than...**, specify the threshold value.

9. Choose **Additional configuration**. For **Datapoints to alarm**, specify how many evaluation periods (data points) must be in the ALARM state to trigger the alarm. If the two values here match, you create an alarm that goes to ALARM state if that many consecutive periods are breaching.

   To create an M out of N alarm, specify a lower number for the first value than you specify for the second value. For more information, see Alarm evaluation.

10. For **Missing data treatment**, choose how to have the alarm behave when some data points are missing. For more information, see Configuring how CloudWatch alarms treat missing data.

11. Choose **Next**.

12. Under **Notification**, select an SNS topic to notify when the alarm is in ALARM state, OK state, or INSUFFICIENT_DATA state.

    To have the alarm send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

    To have the alarm not send notifications, choose **Remove**.

13. To have the alarm perform Auto Scaling, EC2, Lambda, or Systems Manager actions, choose the appropriate button and choose the alarm state and action to perform. If you choose a Lambda function as an alarm action, you specify the function name or ARN, and you can optionally choose a specific version of the function.

    Alarms can perform Systems Manager actions only when they go into ALARM state. For more information about Systems Manager actions, see see Configuring CloudWatch to create OpsItems from alarms and Incident creation.

    > **Note**
    >
    > To create an alarm that performs an SSM Incident Manager action, you must have certain permissions. For more information, see Identity-based policy examples for AWS Systems Manager Incident Manager.

14. When finished, choose **Next**.
15. Enter a name and description for the alarm. Then choose **Next**.

    The name must contain only UTF-8 characters, and can't contain ASCII control characters. The description can include markdown formatting, which is displayed only in the alarm **Details** tab in the CloudWatch console. The markdown can be useful to add links to runbooks or other internal resources.

16. Under **Preview and create**, confirm that the information and conditions are what you want, then choose **Create alarm**.

Create a Log Alarm on error patterns

You can create a Log Alarm that monitors your logs for error patterns and sends a notification when the error count exceeds a threshold.

The following example creates a Log Alarm that transitions to ALARM state when more than 10 ERROR messages appear in 3 out of 5 query executions.

```
aws cloudwatch put-log-alarm \
    --alarm-name "LambdaErrors" \
    --alarm-description "Alarm when Lambda error count exceeds 10" \
    --comparison-operator GreaterThanThreshold \
    --threshold 10 \
    --query-results-to-evaluate 5 \
    --query-results-to-alarm 3 \
    --treat-missing-data notBreaching \
    --alarm-actions "arn:aws:sns:region:account-id:topic-name" \
    --scheduled-query-configuration '{
        "QueryString": "fields @timestamp, @message | filter @message like /ERROR/",
        "LogGroupIdentifiers": ["/aws/lambda/my-function"],
        "ScheduledQueryRoleARN": "arn:aws:iam::account-id:role/ScheduledQueryRole",
        "AggregationExpression": "count(*)",
        "ScheduleConfiguration": {
            "ScheduleExpression": "rate(5 minutes)",
            "StartTimeOffset": 300
        }
    }'
```

Create a Log Alarm on high latency per endpoint

You can create a multi-contributor Log Alarm that monitors average latency grouped by endpoint. The alarm transitions to ALARM state if any endpoint exceeds the latency threshold.

The following example creates a Log Alarm that enters ALARM state when any endpoint has average latency greater than 1000 ms in 3 out of 5 query executions.

```
aws cloudwatch put-log-alarm \
    --alarm-name "EndpointLatency" \
    --alarm-description "Alarm when any endpoint latency exceeds 1000ms" \
    --comparison-operator GreaterThanThreshold \
    --threshold 1000 \
    --query-results-to-evaluate 5 \
    --query-results-to-alarm 3 \
    --treat-missing-data missing \
    --alarm-actions "arn:aws:sns:region:account-id:topic-name" \
    --scheduled-query-configuration '{
```

```
        "QueryString": "parse @message /endpoint=(?<endpoint>\\S+).*latency=(?<latency>
\\d+)/",
        "LogGroupIdentifiers": ["/aws/apigateway/my-api"],
        "ScheduledQueryRoleARN": "arn:aws:iam::account-id:role/ScheduledQueryRole",
        "AggregationExpression": "avg(latency) by endpoint",
        "ScheduleConfiguration": {
            "ScheduleExpression": "rate(5 minutes)",
            "StartTimeOffset": 300
        }
    }'
```

In this example, each unique value of `endpoint` becomes a contributor. If any contributor's average latency exceeds 1000 ms, the alarm transitions to `ALARM` state and sends a notification identifying which endpoint is breaching.

Create an alarm that triggers an Auto Scaling action

With a metric alarm, you can trigger an Auto Scaling scaling policy to add or remove capacity when a metric breaches a threshold. This example scales an Auto Scaling group when the average CPU of the group exceeds a threshold.

You first create the scaling policy, and then attach its ARN as the alarm action. This applies to simple and step scaling policies. For target tracking scaling policies, the Auto Scaling service creates and manages the alarms for you.

Composite alarms can't trigger Auto Scaling actions. For more information, see Auto Scaling actions.

Setting up an Auto Scaling action alarm using the AWS Management Console

Use the AWS Management Console to create an alarm that triggers an Auto Scaling action. Because the full workflow has more than seven steps, it is divided into two parts.

Part 1: Select the metric and set the condition

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. Choose **Select metric**.

5. Select the **AWS/EC2** namespace, and then select **By Auto Scaling Group**. Find the row with **CPUUtilization** for the **AutoScalingGroupName** dimension set to your group. Select the check box next to this row, and choose **Select metric**.
6. For **Statistic**, choose **Average**, and choose a period (for example, **`5 minutes`**).
7. Under **Conditions**, for **Threshold type** choose **Static**. For **Whenever CPUUtilization is**, choose **Greater**, and under **than...** enter **`70`**. Then choose **Next**.

Part 2: Configure the action and create the alarm

1. Under **Notification**, configure an Amazon SNS notification, or choose **Remove** if you don't want one.
2. Choose **Add EC2 Auto Scaling action**. Select the alarm state that triggers the action, select the Auto Scaling group, and then select the scaling policy to run. Then choose **Next**.
3. Under **Name and description**, enter a name and description for the alarm, and choose **Next**.
4. Under **Preview and create**, choose **Create alarm**.

It might take a few minutes for the alarm to become active.

Setting up an Auto Scaling action alarm using the AWS CLI

Use the AWS CLI to create an alarm that triggers an Auto Scaling action.

To create an alarm that triggers an Auto Scaling action

1. Create a simple scaling policy using the put-scaling-policy command. Replace *`my-asg`* and *`scale-out`* with your Auto Scaling group name and policy name.

   ```
   aws autoscaling put-scaling-policy --auto-scaling-group-name my-asg --policy-
   name scale-out --policy-type SimpleScaling --adjustment-type ChangeInCapacity --
   scaling-adjustment 1 --cooldown 300
   ```

   After the command runs, copy the `PolicyARN` value from the output to use in the next step. The output looks similar to the following:

   ```
   {
       "PolicyARN": "arn:aws:autoscaling:us-
   east-1:111122223333:scalingPolicy:12345678-1234-1234-1234-123456789012:autoScalingGroupName
   my-asg:policyName/scale-out"
   ```

```
}
```

2. Create the alarm using the put-metric-alarm command. Replace *my-asg* with your Auto Scaling group name, and replace the ARN in `--alarm-actions` with the `PolicyARN` value from the previous step.

```
aws cloudwatch put-metric-alarm --alarm-name asg-cpu-high --alarm-description
 "Scale out when CPU exceeds 70%" --metric-name CPUUtilization --namespace
 AWS/EC2 --statistic Average --period 300 --threshold 70 --comparison-operator
 GreaterThanThreshold --dimensions Name=AutoScalingGroupName,Value=my-
asg --evaluation-periods 2 --alarm-actions arn:aws:autoscaling:us-
east-1:111122223333:scalingPolicy:...:policyName/scale-out
```

If the command succeeds, it returns no output.

Telemetry discovery and enablement

Amazon CloudWatch telemetry configuration provides a centralized experience to discover and audit the state of telemetry across your AWS resources, and to create rules that automatically enable telemetry for popular AWS resource types. You can use telemetry configuration within a single account or across multiple accounts in AWS Organizations using the management account or a CloudWatch delegated administrator account.

Topics

- What is telemetry discovery and enablement?
- Setting up telemetry configuration
- Discovering resource telemetry
- Telemetry enablement rules
- Troubleshooting telemetry configuration
- Disabling telemetry configuration

What is telemetry discovery and enablement?

CloudWatch telemetry configuration gives you two core capabilities:

- **Discovery and auditing** – Discover AWS resources across your account or organization and audit which resources have telemetry enabled. The experience shows the configuration status at the resource-type level and at more granular telemetry-detail levels.
- **Enablement rules** – Create rules that automatically configure telemetry collection for AWS resources that match your criteria. Rules help you standardize telemetry collection across your organization or accounts and make sure consistent monitoring coverage.

Telemetry configuration supports the following resource types for **discovery and auditing** (using the `ListResourceTelemetry` and `ListResourceTelemetryForOrganization` APIs):

- Amazon Amazon EC2 Instance (`AWS::EC2::Instance`) – Detailed Metrics
- Amazon Amazon VPC (`AWS::EC2::VPC`) – Flow Logs, Route 53 Resolver Query Logs
- AWS Lambda Function (`AWS::Lambda::Function`) – Active Tracing
- Amazon EKS Cluster (`AWS::EKS::Cluster`) – Control Plane Logs

- AWS WAFv2 Web ACL (`AWS::WAFv2::WebACL`) – WAF Logs
- Elastic Load Balancing Network Load Balancer (`AWS::ElasticLoadBalancingV2::LoadBalancer`) – NLB Access Logs

> **Note**
>
> The `AWS::ElasticLoadBalancingV2::LoadBalancer` resource type for discovery only includes Network Load Balancers (NLBs). Application Load Balancers (ALBs) are not currently supported for discovery.

In addition to the resource types above, telemetry **enablement rules** (using the `CreateTelemetryRule` and `CreateTelemetryRuleForOrganization` APIs) support the following additional data sources:

- AWS CloudTrail Data Events and Management Events
- Amazon Bedrock AgentCore Logs
- AWS Security Hub
- Amazon Bedrock AgentCore Gateway
- Amazon Bedrock AgentCore Memory
- Amazon CloudFront Distribution
- Amazon S3 Server Access Logs
- Amazon MSK Cluster Metrics
- OpenTelemetry Enrichment Metrics
- Amazon Bedrock AgentCore Workload Identity
- Elastic Load Balancing Application Load Balancer Logs
- Amazon Bedrock Knowledge Base Logs
- Amazon Bedrock AgentCore Payment Manager
- AWS X-Ray Transaction Search
- Amazon EKS Container Insights

When you enable telemetry configuration, CloudWatch creates AWS Config service-linked configuration recorders that discover resources and their associated telemetry configuration metadata. For more information, see Configuration Recorder in the AWS Config Developer Guide.

> **Note**
>
> AWS Config periodically takes inventory of, or discovers, all the resources in your account as an anti-entropy behavior, regardless of the resource types in scope for your configuration recorders. The inventory includes deleted resources and resources that AWS Config is not currently recording. This behavior helps maintain data consistency.
> This means that although the service-linked configuration recorder for the CloudWatch telemetry configuration feature is configured to record specific resource types, you might see describe calls from `ConfigResourceCompositionSession` and `AWSConfig-Describe` in AWS CloudTrail. For more information, see Non-recorded Resources in the AWS Config Developer Guide.

Amazon CloudWatch uses AWS Config Internal service linked recorder. You are not charged for CIs that CloudWatch uses as part of the Internal Service Linked Recorders.

You can manage telemetry configuration across multiple AWS Regions from a single Region. When you enable multi-Region support, the current Region becomes your home Region and telemetry configuration is replicated to the Regions you select. For more information, see Setting up telemetry configuration.

Setting up telemetry configuration

Use the CloudWatch console to set up telemetry configuration for your AWS account or organization. For an organization, as a management account or a CloudWatch delegated administrator account, CloudWatch discovers AWS resources and provides visibility into the telemetry configurations across all the member accounts in the organization.

Telemetry config remains active until you turn it off. For more information, see Disabling telemetry configuration.

Topics

- Prerequisites and permissions

- Enable telemetry configuration for your account
- Enable telemetry configuration for your organization
- Registering a delegated administrator account for your organization

Prerequisites and permissions

Telemetry configuration for your organization requires trusted access between AWS Organizations and CloudWatch. When trusted access is enabled, CloudWatch creates a service-linked role named **AWSServiceRoleForObservabilityAdmin** to support resource and telemetry configuration discovery for the organization.

When you turn on telemetry configuration for your organization from a management account, CloudWatch automatically enables trusted access and creates the service-linked role. You do not need to set up trusted access manually before starting telemetry evaluation.

> **Important**
>
> CloudWatch uses your IAM permissions to perform this setup. It does not use CloudWatch service permissions. In addition to the telemetry configuration permissions, you must be allowed to perform the following actions:
>
> - `iam:CreateServiceLinkedRole`—to create the **AWSServiceRoleForObservabilityAdmin** and **AWSServiceRoleForObservabilityAdmin_TelemetryEnablement** service-linked roles.
> - `organizations:EnableAWSServiceAccess`—to enable trusted access between AWS Organizations and CloudWatch.
>
> If you are not allowed to perform these actions, turning on telemetry configuration fails with an `AccessDeniedException`. CloudWatch does not enable trusted access.

The following example JSON policy grants the additional permissions required to turn on telemetry configuration for an organization:

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "AllowCreateObservabilityAdminServiceLinkedRoles",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": [
                "arn:aws:iam::*:role/aws-service-role/observabilityadmin.amazonaws.com/
AWSServiceRoleForObservabilityAdmin",
                "arn:aws:iam::*:role/aws-service-role/telemetry-
enablement.observabilityadmin.amazonaws.com/
AWSServiceRoleForObservabilityAdmin_TelemetryEnablement"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "observabilityadmin.amazonaws.com",
                        "telemetry-enablement.observabilityadmin.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "AllowEnableTrustedAccessForObservabilityAdmin",
            "Effect": "Allow",
            "Action": "organizations:EnableAWSServiceAccess",
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> If you use a delegated administrator account to turn on telemetry configuration, the service-linked role is automatically created in all member accounts but is *not* automatically created in the management account. In this case, you must manually create the **AWSServiceRoleForObservabilityAdmin** service-linked role in the management account.

For more information about the service-linked role, see Service-linked role permissions for CloudWatch telemetry config. For more information about AWS Organizations, see Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

Enable telemetry configuration for your account

Configure telemetry for your AWS account to monitor telemetry for the AWS resources in that account. If you have an organization in AWS Organizations, configure telemetry for your organization instead. For more information, see Configuring telemetry for your organization.

To configure telemetry for your AWS account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. Choose the **Data Source** tab, and then select **Enable Resource Discovery**. CloudWatch begins discovering AWS resources in your account. As CloudWatch discovers resources, it updates information on the **Overview** page.

> **Note**
>
> The delay before resources appear on the **Overview** page depends on the number of resources in your account.

Enabling across Regions

You can extend telemetry configuration to multiple AWS Regions from a single Region. When you enable multi-Region support, the current Region becomes your *home Region*. Telemetry configuration is replicated to the Regions you select.

To enable telemetry configuration across Regions for your account (initial setup)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then choose the **Account** tab.
3. In the **CloudWatch telemetry config** section on the **Global** tab, the status shows **Off**. When multi-Region is enabled, a **Target regions** selector appears inline below the status.
4. Use the **All regions** toggle to include all Regions, or use the multiselect dropdown to choose individual Regions. The current Region is always included automatically and is not shown in the selector.
5. Choose **Turn on**.

6. After telemetry configuration is turned on, a **Region status** table appears showing the per-Region evaluation status.

To reconfigure Regions (telemetry already running)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then choose the **Account** tab.
3. In the **CloudWatch telemetry config** section, choose **Configure regions**. The **Target regions** selector appears inline, pre-populated with the currently configured Regions.
4. Modify the Region selection as needed, and then choose **Save**.

If you select **All regions**, new Regions are automatically included when you opt in to them. The system periodically reconciles configuration across Regions to correct any drift.

Enable telemetry configuration for your organization

To turn on telemetry configuration for your organization, you must use a AWS Organization management account or a delegated administrator account. CloudWatch uses this account to discover your organization's AWS resources and configure their telemetry.

When you turn on telemetry configuration from a management account, CloudWatch automatically sets up trusted access between AWS Organizations and CloudWatch, including creating the required service-linked role. You do not need to enable trusted access manually as a separate step. CloudWatch uses your own IAM permissions to perform this setup. You must be allowed to perform `iam:CreateServiceLinkedRole` and `organizations:EnableAWSServiceAccess`. For more information about permissions, see Prerequisites and permissions.

To turn on telemetry auditing for your organization

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organizations** tab.
4. On the **CloudWatch** settings page, in the **Organizational settings management** pane, choose **Turn on**. CloudWatch automatically enables trusted access and creates the **AWSServiceRoleForObservabilityAdmin** service-linked role in your organization.

To review the role policy, choose **View permission details**.

5. After Telemetry config is turned on for the organization, a notification appears. On the notification, choose Go to Telemetry config. The Telemetry Configuration experience can be accessed in the **Ingestion** page and CloudWatch begins discovering AWS resources in the organization. As CloudWatch discovers resources, it updates information on the **Telemetry config** page.

> **Note**
>
> The time delay before resources appear on the **Telemetry config** page depends on the number of member accounts and resources in your organization or account.

Configuring telemetry for your organization

Configure telemetry for AWS Organizations to monitor the telemetry for the AWS resources across all your member accounts. This also configures the telemetry for individual accounts. You can also configure telemetry for only your account. For more information, see Enable telemetry configuration for your account.

You can disable trusted access across all your member accounts. For more information, see Turning off trusted access for AWS Organizations.

To configure telemetry auditing for your organization

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. Choose **Data sources**, and then choose the **Enable Resources Discovery Button**. CloudWatch begins discovering AWS resources in your organization. As CloudWatch discovers resources, it updates information in the **Overview** page.

> **Note**
>
> The delay before resources appear on the **Overview** page depends on the number of member accounts and resources in your organization.

Enabling across Regions

You can extend telemetry configuration to multiple AWS Regions from a single Region for your entire organization. When you enable multi-Region support, the current Region becomes your *home Region*. Telemetry configuration is replicated to the Regions you select for all member accounts.

To enable telemetry configuration across Regions for your organization (initial setup)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then choose the **Organizations** tab.
3. In the **CloudWatch telemetry config** section on the **Global** tab, the status shows **Off**. When multi-Region is enabled, a **Target regions** selector appears inline below the status.
4. Use the **All regions** toggle to include all Regions, or use the multiselect dropdown to choose individual Regions. The current Region is always included automatically and is not shown in the selector.
5. Choose **Turn on**.
6. After telemetry configuration is turned on, a **Region status** table appears showing the per-Region evaluation status.

To reconfigure Regions for your organization (telemetry already running)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then choose the **Organizations** tab.
3. In the **CloudWatch telemetry config** section, choose **Configure regions**. The **Target regions** selector appears inline, pre-populated with the currently configured Regions.
4. Modify the Region selection as needed, and then choose **Save**.

If you select **All regions**, new Regions are automatically included when you opt in to them. The system periodically reconciles configuration across Regions to correct any drift.

Registering a delegated administrator account for your organization

A delegated administrator account is a member account that shares administrator access for service-managed permissions. The account that you register as a delegated administrator must be in your organization. A delegated administrator account for your organization can be used

outside of CloudWatch, so make sure that you understand this account type before you follow this procedure. For more information, see Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

To remove or change the delegated administrator account, deregister the account first. For more information, see Deregistering a delegated administrator account.

To register a delegated administrator account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. In the **Organizational settings management** pane, choose **Register delegated administrator**.
5. In the **Register delegated administrator** dialog, for **Delegated administrator account ID**, enter the 12-digit account ID for an organization member account.
6. Choose **Register delegated administrator**. At the top of the **CloudWatch settings** page, a message appears indicating the account was registered successfully. To see information about the delegated administrator account, select the number below **Delegated administrators**.

Deregistering a delegated administrator account

Deregister the delegated administrator account before turning off trusted access for AWS Organizations. You can also deregister a delegated administrator account if it no longer has access to the appropriate AWS resources for telemetry configuration or to choose a different member account to be the delegated administrator. This account will not be able to perform account management tasks for AWS Organizations. For more information, see Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

To deregister the delegated administrator account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. On the **Organization** tab, choose **Deregister**.
4. On the **Deregister delegated administrator** page, choose **Deregister**.

To register an account as a delegated administrator, see Registering a delegated administrator account for your organization.

Turning off trusted access for AWS Organizations

Trusted access extends the functionality of the management account in AWS Organizations to other AWS services. When you turn off trusted access, trusted access between your organization and all AWS services—not just CloudWatch—will stop.

If you no longer want trusted access turned on for your organization, you can turn it off. For more information, see Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

> **Note**
>
> Before turning off trusted access for an organization, deregister the delegated administrator account. For more information, see Deregistering a delegated administrator account.

To turn off trusted access for AWS Organizations

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. In the **Organizational Management Settings** section, select **Turn off**.

Discovering resource telemetry

The telemetry configuration experience displays AWS resources in two places: as an overview on the **Ingestion – Data sources** and in detail on the **Discovered resources** page.

Topics

- Viewing data sources
- Viewing discovered resources
- Filtering and preferences

Viewing data sources

The **Ingestion – Data sources** shows the AWS resources that you can send to CloudWatch. For specific resource types, it shows the percentage of resources with telemetry configured and the total number of resources detected. You can filter the display of resources in the **Data source** tab by account ID or by the tags applied to your resources.

To view resources on the Ingestion page

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. The **Ingestion** page shows the total number of each resource that was discovered by CloudWatch, the number of resources providing telemetry, and the percentage of discovered resources that are providing telemetry.
4. To see recent changes to resources, choose **Refresh**.

Viewing discovered resources

The **Discovered resources** page shows details about each AWS resource that has been discovered by telemetry configuration, including the resource ID, the type of telemetry each resource is providing, and the time when information about the resource was last refreshed.

For each AWS resource tracked by CloudWatch, the **Discovered resources** page shows the status of its telemetry by providing the following information:

- For telemetry types that CloudWatch detects that the resource is sending to CloudWatch, the **Discovered resources** page shows **On**.
- For telemetry types that CloudWatch detects the resource is not providing, the **Discovered resources** page shows **Off**.
- For telemetry types that are not supported for a resource, the **Discovered resources** page shows **NS**, that is, not supported.

To view resources on the Discovered resources page

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.

3. Do one of the following to view all resource types discovered by telemetry configuration or to view one resource type:

   a. To view all resources that have been discovered by CloudWatch, click **View data sources**. The **Discovered resources** page appears and shows all resources discovered.

   b. To view one resource type, click the name of the AWS resource type in the **Ingestion > Data sources**. The **Discovered resources** page appears. The **Discovered resources** page shows that a filter has been applied for that data source and now displays all discovered resources for that resource type.

4. On the **Discovered resources** page, to view information about the resource or to change its telemetry settings, click the resource ID. The console page for the AWS resource will open, navigate to the configuration setting and turn on or off the configuration.

> **Note**
>
> You can only view a resource on its console page if the resource belongs to your account. To determine if the resource belongs to your account, check the **AWS account** column. If the **AWS account** column does not appear, change your **Discovered resources** page preferences. For more information, see Changing preferences for the Discovered resources page.

Viewing resources across Regions

When multi-Region support is active, the **Discovered resources** page includes a **Region** column. You can filter resources by Region to focus on specific Regions.

CloudWatch uses a AWS Config aggregator to collect resource data across Regions. Because of this, there may be a short delay before resources from spoke Regions appear in the home Region view.

Filtering and preferences

You can use one or more filters on the **Ingestion > Data Sources** page and the **Discovered resources** page to change your view of the resources. Your filter settings persist across both pages.

To filter resources on the Data Sources page

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**, and then choose **Data sources**.

3. You can filter the discovered resources that are displayed on the page by specifying an account ID or tag value.

   a. Choose **Find resource**.

   b. Choose **Account ID** or **Tag value**, and then choose additional options for the filter. Statistics about telemetry coverage for each resource change based on your filter options.

4. To remove a filter, in the filter text box, choose **X**.

To filter resources on the Discovered Data sources page

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. To view all resource types discovered by telemetry configuration or to view one resource type, do one of the following:

   a. To view all resources discovered by CloudWatch, choose **View data sources**. The **Discovered resources** page appears and shows all resources discovered.

   b. To view one resource type, at the bottom of the page, choose a type of AWS resource. The **Discovered resources** page appears. The **Discovered resources** page filters all discovered resources for that resource type.

4. You can filter the resources displayed in the page based on any of the columns in the page. You can change the columns in the page by changing your preferences for the **Discovered resources** page. For more information, see the preferences procedure below.

   a. Choose **Find resource**. Filters for each column in the page appear. Choose one, then choose additional options to define the filter. Resources appear in the page that match the filter settings.

   b. To further filter the resources displayed in the page, choose **Find resources** again, choose another filter, and choose additional options. Resources appear in the page that match all of the filter settings.

5. To remove one of the filters, in the filter text box, choose **X**.
6. To remove all of the filters and see all resource types discovered, choose **Clear filters**.

Changing preferences for the Discovered resources page

You can change your preferences for the **Discovered resources** page to control how many resources appear per page and which detailed metrics appear in the page. Only detailed metrics in view can be used to filter the resources displayed in the discovered resources page. For more information, see Filtering and preferences.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion** and **Data sources tab**.
3. Choose **Discovered resources**. The **Discovered resources** page appears.
4. Choose the gear icon.
5. In the **Preferences** dialog box, choose the number of resources per page and the visible content to show as columns.
6. Choose **Confirm**.

Telemetry enablement rules

You can create telemetry enablement rules to automatically configure telemetry collection for your AWS resources. Rules help you standardize telemetry collection across your organization or accounts and make sure consistent monitoring coverage.

Topics

- How rules work
- Creating a telemetry enablement rule
- Managing telemetry rules
- Encrypting log groups with customer managed keys
- Supported data sources

How rules work

Telemetry configuration follows specific patterns when evaluating and applying rules.

Rule evaluation hierarchy

Enablement rules are evaluated according to a hierarchical pattern. Organizational rules are evaluated first, then rules that apply to organizational units (OUs), and finally rules that apply to

individual accounts. Rules at the organizational level provide the baseline required telemetry for your organization. Rules at the OU and account level can collect additional telemetry data, but they cannot collect less telemetry data. If such a rule is created, it will create a rule conflict.

Within each scope (organization, OU, or account), rules must maintain uniqueness based on their resource type, telemetry type, and destination configuration. Duplicate rules trigger a conflict exception. If the same rule exists in different scopes, such as an organization level rule for Amazon VPC Flow logs to CloudWatch and an OU level rule for Amazon VPC Flow logs, the rule higher in the hierarchy is applied. However, if there are multiple conflicting rules, none of the rules are applied.

When multiple rules apply to the same resource, telemetry configuration resolves conflicts using these priorities:

1. Organizational-level rules take precedence over account-level rules
2. More specific tag matches take precedence over general rules
3. If there are multiple conflicting rules, none of the rules are applied. You must resolve the conflicts first.

Rule behavior on updates

If you update an enablement rule, only new resources that match the rule adopt the updated configuration. The existing telemetry settings remain unchanged for existing resources. If a resource becomes non-compliant with an existing rule due to manual deletion of telemetry data, the new enablement rule is adopted after the resource is brought back into compliance.

For Amazon VPC Flow logs, telemetry config only creates new flow logs for resources that match the rule scope. It does not delete or impact previously established Amazon VPC Flow logs, even if they differ from current rule parameters. For CloudWatch Logs, existing log groups are maintained provided they match the resource pattern.

Integration with AWS Config

CloudWatch telemetry auditing and configuration integrates with AWS Config to automatically discover resources that match your enablement rule and apply it to your telemetry data collection. When you create an enablement rule, the telemetry configuration creates a corresponding AWS Config recorder. This recorder includes configuration items for the specific resource types you define in the enablement rule.

Amazon CloudWatch uses AWS Config Internal service linked recorder. You are not charged for CIs that CloudWatch uses as part of the Internal Service Linked Recorders.

> **Note**
>
> When you create an enablement rule, we discover non-compliant resources (those without telemetry enabled) through AWS Config Configuration Items (CIs) before turning them on based on your enablement rule scope. The initial discovery of the resources may take up to 24 hours to complete in some cases.

Telemetry config uses AWS Config to:

- Discover resources across your organization or accounts
- Track telemetry configuration changes

Rules across Regions

When you create a rule with target Regions, the current Region becomes the *home Region* for that rule. The rule is automatically replicated to the spoke Regions you select.

Key concepts for multi-Region rules:

- Replicated rules cannot be edited or deleted in spoke Regions. You must navigate to the home Region to modify or remove them.
- If you select **All regions**, new Regions are automatically included when you opt in to them.
- The system periodically reconciles rules across Regions to correct any drift between the home Region and spoke Regions.
- Tags applied to rules in the home Region are replicated to spoke Regions.

When a replicated rule is created, updated, or deleted in a spoke Region, AWS CloudTrail records an `AwsServiceEvent` in the spoke Region. These events are logged with `observabilityadmin.amazonaws.com` as the invoking service and include the rule ARN in the spoke Region. You can use these events to audit multi-Region rule replication activity.

The following is an example AWS CloudTrail event recorded when a replicated rule is created in a spoke Region:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "observabilityadmin.amazonaws.com"
    },
    "eventTime": "2026-04-06T19:50:37Z",
    "eventSource": "observabilityadmin.amazonaws.com",
    "eventName": "CreateTelemetryRule",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "observabilityadmin.amazonaws.com",
    "userAgent": "observabilityadmin.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "435d6da2-d099-4775-8944-1e039418de6f",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::ObservabilityAdmin::TelemetryRule",
            "ARN": "arn:aws:observabilityadmin:us-east-1:123456789012:telemetry-
rule/my-multi-region-rule"
        }
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


The `eventName` field reflects the operation performed on the replicated rule: `CreateTelemetryRule`, `UpdateTelemetryRule`, or `DeleteTelemetryRule`. The `eventType` is always `AwsServiceEvent` because the operation is performed by the ObservabilityAdmin service on behalf of the customer, not by a direct customer API call.

Creating a telemetry enablement rule

When you create a telemetry enablement rule, you specify:

- The scope of the rule (organization, organizational unit, or account)
- The resource types the rule applies to

- The telemetry types to enable (metrics, logs, or traces)
- Optional tags to filter which resources the rule affects
- Optional target Regions to replicate the rule across multiple Regions
- Optional AWS KMS key ARN to encrypt log groups created by the rule with a customer managed key

To create a telemetry enablement rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. Choose the **Enablement rules** tab.
4. Choose **Add rule**.
5. For **Rule name**, enter a name for your rule.
6. For **Rule scope**, choose one of the following:
   - **Organization** – Rule applies across your entire AWS Organizations
   - **Organizational unit** – Rule applies to a specific OU
   - **Account** – Rule applies to a single account
7. For **Data source**, select the AWS service to configure.
8. For **Telemetry type**, select the types of telemetry to enable.
9. (Optional) Add tags to filter which resources the rule affects.
10. (Optional) For **Target regions**, select the Regions where you want this rule to apply. The current Region is automatically designated as the home Region for the rule. If you select **All regions**, new Regions are automatically included when you opt in to them.
11. (Optional) For **KMS key ARN**, enter the ARN of an AWS KMS key to encrypt log groups created by this rule. For cross-Region rules, you must use a multi-Region key (key ID starts with `mrk-`). For more information, see Encrypting log groups with customer managed keys.
12. Choose **Create rule**.

Managing telemetry rules

After creating rules, you can edit or delete them. You can also view which resources each rule affects and monitor rule compliance.

To manage an existing rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Ingestion**.
3. Choose the **Enablement rules** tab.
4. Select a rule to view its details or choose one of these actions:

   - **Edit rule** – Modify rule settings
   - **Delete** – Remove the rule

Managing replicated rules

When you view a replicated rule in a spoke Region, the console displays an informational alert indicating that the rule was replicated from another Region. The **Edit rule** and **Delete** actions are disabled for replicated rules in spoke Regions.

To edit or delete a replicated rule, navigate to the home Region where the rule was originally created. The home Region is displayed in the informational alert.

You can add or modify tags on replicated rules in spoke Regions. Tag changes made in spoke Regions apply only to the local copy of the rule and are not replicated back to the home Region.

Encrypting log groups with customer managed keys

You can encrypt log groups created by telemetry enablement rules using a customer managed AWS KMS key. Encrypting log groups with a customer managed key gives you control over key rotation and helps meet compliance requirements. When you specify a AWS KMS key ARN in your rule's destination configuration, CloudWatch automatically associates the key with each log group created during remediation.

Requirements for KMS keys

- For cross-Region rules, you must use a multi-Region AWS KMS key. Multi-Region keys have a key ID that starts with `mrk-`. This ensures consistent encryption across all Regions where the rule is applied.
- For rules that target a single Region, both single-Region and multi-Region keys are accepted.
- The AWS KMS key must be enabled and accessible to the service-linked role used by telemetry rules.

- When you create or update a rule with a AWS KMS key ARN, the service validates that the key exists and is enabled by calling `kms:DescribeKey`.

Required KMS key policy

Your AWS KMS key policy must allow the CloudWatch Logs service to use the key for encryption. Add the following statement to your key policy:

```
{
    "Sid": "AllowCloudWatchLogsToUseKey",
    "Effect": "Allow",
    "Principal": { "Service": "logs.amazonaws.com" },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceOrgID": "your-organization-id"
        },
        "ArnLike": {
            "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:*:*:log-group:*"
        }
    }
}
```

Replace *your-organization-id* with your AWS Organizations organization ID. The `aws:SourceOrgID` condition ensures that only accounts in your organization can use the key for log group encryption.

> **Note**
>
> For organization-scope rules that remediate across member accounts, this key policy grants the CloudWatch Logs service in member accounts permission to encrypt and decrypt log data using your key. The telemetry rules service-linked role does not directly perform encryption or decryption—it only associates the key with the log group.

Service-linked role permissions for encryption

The telemetry rules service-linked role requires additional permissions to support AWS KMS key encryption. The following permissions are automatically added to the service-linked role when you use AWS KMS encryption with telemetry rules:

- `kms:DescribeKey` – Allows the service to validate that the AWS KMS key exists, is enabled, and is a multi-Region key (for cross-Region rules). This permission is used during rule creation and update validation. The call is always same-account (the service-linked role describes the key in the rule creator's own account).
- `logs:AssociateKmsKey` – Allows the service to associate the AWS KMS key with log groups created during remediation. This permission is scoped to log groups that are tagged with `CloudWatchTelemetryRuleManaged: true`, which limits the association to log groups managed by telemetry rules.

> **Note**
>
> The service-linked role does not perform encryption or decryption of log data directly. After the service associates the AWS KMS key with a log group, CloudWatch Logs uses the key for all subsequent encrypt and decrypt operations on that log group. Cross-account AWS KMS access during remediation is handled by the AWS KMS key policy (granting the `logs.amazonaws.com` service principal access), not by the service-linked role.

How multi-Region keys work with telemetry rules

Multi-Region AWS KMS keys share the same key ID across Regions. When a telemetry rule with a AWS KMS key is applied in multiple Regions, the service automatically resolves the key ARN to the target Region. For example, if you provide the key ARN `arn:aws:kms:us-east-1:123456789012:key/mrk-1234abcd` and the rule creates a log group in `eu-west-1`, the service uses `arn:aws:kms:eu-west-1:123456789012:key/mrk-1234abcd` for encryption in that Region.

You must make sure that the multi-Region key is replicated to all Regions where the rule applies. If the key has not been replicated to a target Region, remediation for resources in that Region fails, and the service retries the operation.

Updating encryption settings

When you update a rule to add a AWS KMS key, the service applies the key only to log groups that the rule creates after the update. The service does not retroactively encrypt log groups that the rule created before you added the key.

When you remove the AWS KMS key ARN from a rule, the service changes the encryption configuration that it previously applied to the rule's managed log groups.

Supported data sources

The following data sources are supported by telemetry enablement rules. Each data source has specific behavior and configuration considerations.

Amazon VPC Flow Logs

When creating flow logs:

- Uses default pattern /aws/vpc/vpc-id if none specified
- Existing customer-created flow logs are preserved
- Rule updates only affect new flow logs
- You can use <vpc-id>, <account-id> macros to split log groups.
- CloudWatch does not create flow logs for VPCs that already are ingesting logs to CloudWatch Logs
- When you enable automatic configuration updates on a rule, CloudWatch monitors the flow logs it created and remediates configuration drift. Drift occurs when the log format, traffic type, maximum aggregation interval, or destination log group name pattern of a rule-managed flow log no longer matches the rule. To remediate, CloudWatch creates a new flow log with the correct configuration and then deletes the outdated one.
- Automatic configuration updates apply only to Amazon VPC Flow Logs. CloudWatch never modifies or deletes flow logs that you created.

Amazon EKS Control Plane Logs

When enabling control plane logging:

- Uses default CloudWatch log group pattern /aws/eks/<cluster-name>/cluster. Amazon EKS creates Log Group per Cluster automatically.

- Rule updates only affect new clusters or only clusters that do not have the scoped log types enabled
- Can enable specific log types: api, audit, authenticator, controllerManager, scheduler

Amazon EKS Container Insights

When enabling Container Insights:

- Supports the INSIGHTS telemetry type on the `AWS::EKS::Cluster` resource type. The rule enables Container Insights as a whole and ignores any telemetry source types that you specify.
- Does not support a destination configuration. CloudWatch installs the Amazon CloudWatch Observability EKS add-on on each matching cluster, and the add-on sends metrics and logs to CloudWatch. For more information, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.
- CloudWatch creates one service role named **CloudWatchTelemetryRuleEKSAddonRole** that every matching cluster in the account shares. CloudWatch always creates this role from an AWS managed IAM role template, so you cannot direct CloudWatch to use a different role or a different configuration.
- CloudWatch identifies the role it created by the `CloudWatchTelemetryRuleManaged` tag. If your account already has a role with this name that does not have that tag, CloudWatch does not install the add-on on any cluster in the account, and does not report an error. Rename or delete the conflicting role, and CloudWatch installs the add-on on the next evaluation.
- You can create only one Container Insights rule per account or organization in each Region. This limit does not apply to Amazon EKS control plane logs rules.
- Deleting the rule does not stop telemetry collection or the associated charges. To stop collection, uninstall the add-on from each cluster, which also removes the add-on-managed Amazon EKS Pod Identity association, and then delete the IAM role if no cluster still needs it.
- Available only in the commercial AWS Regions where CloudWatch telemetry configuration is available. Not available in the AWS GovCloud (US) Regions or in the European Sovereign Cloud. In other Regions, `CreateTelemetryRule` returns a validation error and the option does not appear in the console.

AWS WAF Web ACL Logs

When creating WAF logs:

- Uses default CloudWatch log group pattern and always prefixes with aws-waf-logs-
- Rule updates only affect new Web ACLs or existing Web ACLs that do not have logging enabled to CloudWatch Logs
- CloudWatch does not enable logs for Web ACLs that already are ingesting logs to CloudWatch Logs

Amazon Route 53 Resolver Logs

When enabling resolver query logging:

- Uses default CloudWatch log group pattern /aws/route53resolver if none specified
- You can use <account-id> macros to split log groups.
- CloudWatch does not create resolver query logs for VPCs that already are ingesting logs to CloudWatch Logs
- Enablement rules configure Route 53 query logging for your VPCs based on rule scope. CloudWatch does not discover Route 53 profiles and related configurations.

NLB Access Logs

When enabling access logs:

- Uses default CloudWatch log group pattern with prefix /aws/nlb/access-logs if none specified
- CloudWatch does not enable log deliveries for NLBs that already are ingesting logs to CloudWatch Logs

CloudTrail Logs using service-linked channel

When enabling CloudTrail logs using the SLC path:

- Uses managed CloudWatch log groups aws/cloudtrail/<event-types>
- Existing customer-created CloudTrail Trail forwarding configurations are preserved
- CloudWatch Enablement Rules only uses service-linked channel to ingest logs
- Events use the retention period configured for the log group
- For CloudTrail events, as part of the enablement wizard, you can choose at least one event type to ingest to CloudWatch.
- If events are delivered with delay (indicated by addendum reason DELIVERY_DELAY) and you previously configured a shorter retention period, delayed events might only be available for the duration of the shorter retention period.

> **Tip**
>
> To configure CloudTrail logs across multiple Regions, use the **Target regions** selector when creating your enablement rule. This replicates the rule to your selected Regions automatically from the home Region.

Amazon Amazon EC2 Detailed Metrics

When enabling detailed monitoring:

- Instance state changes may affect metric collection

AWS Security Hub

When enabling Security Hub logging:

- Uses managed CloudWatch log group pattern aws/securityhub_cspm/findings
- CloudWatch does not enable log deliveries for Security Hub that already are ingesting logs to managed CloudWatch Logs

Amazon Bedrock AgentCore

- Enable both logs and traces emitted from all available Bedrock AgentCore primitives such as Runtime, Browser Tools, Code Interpreter Tools, and so on. Follow the Telemetry Configure console experience for creating a logs delivery rule then followed by creating a traces delivery rule.
- When creating a trace delivery rule, Transaction Search will be enabled and additional permission policy will be created to allow for CloudWatch X-Ray to send correlated trace to managed log group in your account. In addition, X-Ray resource policy will be created to allow for current and new Bedrock AgentCore primitives to deliver traces to your account.

Amazon Bedrock Agentcore Gateway

When enabling Bedrock Agentcore Gateway logging:

- Uses default CloudWatch log group pattern /aws/bedrock/agentcore if none specified
- CloudWatch does not enable log deliveries for Bedrock Agentcore Gateway that already are ingesting logs to CloudWatch Logs

Amazon Bedrock Agentcore Memory

When enabling Bedrock Agentcore Memory logging:

- Uses default CloudWatch log group pattern /aws/bedrock/agentcore if none specified

- CloudWatch does not enable log deliveries for Bedrock Agentcore Memory that already are ingesting logs to CloudWatch Logs

Amazon CloudFront Distribution

When enabling CloudFront Distribution logging:

- CloudWatch does not enable log deliveries for CloudFront distributions that already are ingesting logs to CloudWatch Logs

Amazon S3 Server Access Logs

S3 server access logging has the following constraints:

- Supports the LOGS telemetry type with log type S3_SERVER_ACCESS_LOGS only.
- Supports only CloudWatch Logs as the destination type.
- Supports only tag-based selection criteria to target specific S3 buckets.

Amazon MSK Cluster Metrics

When enabling MSK Cluster metrics:

- Only supports METRICS telemetry type
- You can configure enhanced monitoring levels (PER_BROKER, PER_TOPIC_PER_BROKER, and so on) to control the granularity of metrics collected
- Rules with different enhanced monitoring levels can coexist for the same MSK cluster

OpenTelemetry Enrichment Metrics

When enabling OpenTelemetry Enrichment metrics:

- Only supports METRICS telemetry type
- This is an account-level enablement with no user-configurable destination
- Resource-level selection criteria is not supported

Amazon Bedrock Agentcore Workload Identity

When enabling Bedrock Agentcore Workload Identity logging:

- Uses default CloudWatch log group pattern /aws/bedrock/agentcore if none specified
- CloudWatch does not enable log deliveries for Bedrock Agentcore Workload Identity that already are ingesting logs to CloudWatch Logs

Elastic Load Balancing Application Load Balancer Logs

When enabling Application Load Balancer logging:

- Supports the LOGS telemetry type with log types `ALB_ACCESS_LOGS`, `ALB_CONNECTION_LOGS`, and `ALB_HEALTH_CHECK_LOGS`.
- Supports only CloudWatch Logs as the destination type.
- CloudWatch does not enable log deliveries for Application Load Balancers that already are ingesting the specified log types to CloudWatch Logs

Amazon Bedrock Knowledge Base

When enabling Bedrock Knowledge Base telemetry:

- Supports the LOGS telemetry type with log type `APPLICATION_LOGS`.
- Supports the TRACES telemetry type.
- For LOGS, supports only CloudWatch Logs as the destination type.
- CloudWatch does not enable log deliveries for Bedrock Knowledge Bases that already are ingesting the specified log types to CloudWatch Logs

Amazon Bedrock AgentCore Payment Manager

When enabling Bedrock AgentCore Payment Manager telemetry:

- Supports the LOGS telemetry type with log type `APPLICATION_LOGS`. Payment Manager does not support `USAGE_LOGS`.
- Supports the TRACES telemetry type.
- For LOGS, supports only CloudWatch Logs as the destination type.
- CloudWatch does not enable log deliveries for payment managers that already are ingesting the specified log types to CloudWatch Logs

AWS X-Ray Transaction Search

When enabling Transaction Search:

- Supports the TRACES telemetry type on the `AWS::XRay::TransactionSearchConfig` resource type. This resource type is the mechanism that enables Transaction Search, so telemetry source types do not apply and the rule ignores any that you specify. For more information, see Transaction Search.
- Transaction Search is an account-wide setting. Enabling it switches all span ingestion for the account through X-Ray into CloudWatch Logs, which changes how that span data is billed.
- Does not support a destination configuration. Transaction Search always sends spans to the service-managed `aws/spans` log group.

- CloudWatch creates the `aws/spans` and `/aws/application-signals/data` log groups if they do not already exist, and attaches a resource policy to each one that allows X-Ray to call `logs:PutLogEvents` on it.
- If either log group already exists and CloudWatch telemetry rules did not create it, CloudWatch does not modify the log group or its resource policy.
- CloudWatch does not change your span indexing configuration. Transaction Search uses whatever indexing rate the account already has.
- CloudWatch does not change accounts that already have Transaction Search enabled.
- You can create only one Transaction Search rule per scope. An account-level rule and an organization-level rule are tracked separately, so both can exist at the same time and both act on the same account-wide setting.
- Deleting the rule does not turn Transaction Search off, and does not delete the log groups or their resource policies. To turn Transaction Search off, set the X-Ray trace segment destination back to X-Ray yourself.

Troubleshooting telemetry configuration

This section describes common issues you might encounter when using telemetry configuration and how to resolve them.

Resources not appearing

If resources are not appearing in discovery, verify the following:

- The resource type is supported by telemetry configuration. For a list of supported data sources, see Supported data sources.
- The AWS Config recorder is enabled in your account. Telemetry configuration requires AWS Config service-linked recorders to discover resources.
- You have appropriate IAM permissions to view the resources.
- Sufficient time has elapsed since enabling telemetry configuration. The initial discovery of resources may take up to 24 hours to complete in some cases.

Rules not applying

If enablement rules are not applying to your resources, check the following:

- Verify the rule scope configuration. Make sure the rule targets the correct organization, OU, or account.
- Check tag filters. If the rule uses tag-based filtering, verify that the target resources have the expected tags.
- Check for rule conflicts. If multiple conflicting rules exist, none of the conflicting rules are applied. For more information, see Rule evaluation hierarchy.
- For Amazon EKS Container Insights, check the cluster state. A rule does not apply to a cluster until the cluster reaches the `ACTIVE` state. CloudWatch retries on a later evaluation cycle.
- For Amazon EKS Container Insights, check the add-on state. The Amazon CloudWatch Observability EKS add-on deploys the CloudWatch agent as a DaemonSet, so on a cluster that has no nodes the add-on installs but reports `DEGRADED` in the Amazon EKS `DescribeAddon` response and sends no telemetry. CloudWatch treats the rule as satisfied and does not retry. The add-on starts sending telemetry when a node becomes available.

> **Note**
>
> When you create an enablement rule, we discover non-compliant resources (those without telemetry enabled) through AWS Config Configuration Items (CIs) before turning them on based on your enablement rule scope. The initial discovery of the resources may take up to 24 hours to complete in some cases.

Multi-Region issues

When using multi-Region telemetry configuration, you might encounter the following issues:

- **Spoke Region failures** – If a rule fails to replicate to a spoke Region, the failure is visible in the rule status dashboard. The system automatically retries failed replications. Check the per-Region status in the console or by using the API to identify which Regions are affected.
- **Delayed resource discovery** – Resources from spoke Regions may take longer to appear in the home Region view because CloudWatch uses a AWS Config aggregator to collect resource data across Regions.
- **Reconciliation drift** – The system periodically reconciles rules across Regions to correct drift. If you notice inconsistencies between the home Region and spoke Regions, allow time for the reconciliation process to complete.

Home Region conflicts

You might encounter errors when attempting to edit or delete a rule from a Region that is not the home Region for that rule. Replicated rules can only be modified in the home Region where they were originally created.

To resolve home Region conflicts:

- Check the informational alert displayed on the rule in the console. The alert identifies the home Region for the rule.
- Navigate to the home Region in the CloudWatch console to edit or delete the rule.
- If you need to create a different rule in the spoke Region, create a new rule with a different name and scope that does not conflict with the replicated rule.

Disabling telemetry configuration

When you no longer need telemetry configuration, you can turn it off. When telemetry configuration is turned off, it no longer shows the status of telemetry for resources in your account or organization. You can turn on telemetry configuration again at any time. For more information, see Setting up telemetry configuration.

To turn off telemetry configuration

1. Open the CloudWatch console.
2. In the navigation pane, choose **Ingestion**.
3. Choose **Turn off**.

> **Note**
>
> Turning off telemetry configuration does not delete or modify any existing telemetry settings for your resources. It only stops the centralized management and visibility of these settings.

Infrastructure monitoring

The topics in this section explain CloudWatch features that can help you gain operational visibility into your AWS resources.

Topics

- Container Insights
- Lambda Insights
- CloudWatch Database Insights
- SageMaker AI Insights
- ElastiCache Insights
- Use Contributor Insights to analyze high-cardinality data
- Detect common application problems with CloudWatch Application Insights
- Using the resource health view in the CloudWatch console

Container Insights

Use CloudWatch Container Insights to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), RedHat OpenShift on AWS (ROSA), and Kubernetes platforms on Amazon EC2. Container Insights supports collecting metrics from clusters deployed on AWS Fargate for both Amazon ECS and Amazon EKS.

CloudWatch automatically collects metrics for many resources, such as CPU, memory, disk, and network. Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. You can also set CloudWatch alarms on metrics that Container Insights collects.

Container Insights collects data as *performance log events* using embedded metric format. These performance log events are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale. From this data, CloudWatch creates aggregated metrics at the cluster, node, pod, task, and service level as CloudWatch metrics. The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and are also viewable in the **Metrics** section of the CloudWatch console. Metrics are not visible until the container tasks have been running for some time.

When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

To help you manage your Container Insights costs, CloudWatch does not automatically create all possible metrics from the log data. However, you can view additional metrics and additional levels of granularity by using CloudWatch Logs Insights to analyze the raw performance log events.

With the original version of Container Insights, metrics collected and logs ingested are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics and logs are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

For Amazon EKS, OTel Container Insights collects metrics using the OpenTelemetry Protocol (OTLP) and supports PromQL queries. Each metric is enriched with up to 150 labels, including OpenTelemetry semantic convention attributes and Kubernetes pod and node labels. For more information, see OTel Container Insights (Recommended).

In Amazon EKS, RedHatOpenshift on AWS, and Kubernetes, Container Insights uses a containerized version of the CloudWatch agent to discover all of the running containers in a cluster. It then collects performance data at every layer of the performance stack.

Container Insights supports encryption with the AWS KMS key for the logs and metrics that it collects. To enable this encryption, you must manually enable AWS KMS encryption for the log group that receives Container Insights data. This causes Container Insights to encrypt this data using the provided KMS key. Only symmetric keys are supported. Do not use asymmetric KMS keys to encrypt your log groups.

For more information, see Encrypt Log Data in CloudWatch Logs Using AWS KMS.

Topics

- Amazon EKS
- Amazon ECS
- Setting up Container Insights on Red Hat OpenShift on AWS (ROSA)
- Viewing Container Insights metrics
- Metrics collected by Container Insights

Amazon EKS

Container Insights on Amazon EKS provides comprehensive observability for your Kubernetes workloads. Two approaches are available: OTel Container Insights (recommended) and Enhanced Container Insights (Classic). Both use the `amazon-cloudwatch-observability` EKS add-on with different configurations.

Topics

- OTel Container Insights (Recommended)
- Enhanced Container Insights (Classic)
- Compare Container Insights approaches
- Migration guides
- Troubleshooting Container Insights on Amazon EKS

OTel Container Insights (Recommended)

OTel Container Insights is the recommended approach for enabling Container Insights on Amazon EKS clusters. Built on OpenTelemetry, it provides comprehensive observability across metrics and logs with a simplified deployment model and active ongoing development.

Both OTel Container Insights and Enhanced Container Insights (Classic) use the same `amazon-cloudwatch-observability` Amazon EKS add-on. The difference is the add-on version and configuration, not a different product. OTel Container Insights is the newer, actively developed configuration of that add-on.

This approach is designed for the following use cases:

- New Amazon EKS customers who want the best-supported path to cluster observability
- Existing customers looking to consolidate on a single, modern telemetry pipeline
- Teams adopting OpenTelemetry who want native integration with CloudWatch

Key benefits

OTel Container Insights provides the following benefits:

- **Simplified setup** — Enable full observability in minutes through the console, AWS CLI, or CloudFormation

- **Comprehensive signals** — Collect metrics and logs from a single agent deployment
- **Enhanced observability** — Access detailed Kubernetes metrics, pod-level insights, and correlated signals
- **Active development** — Receives new features, performance improvements, and expanded signal coverage on an ongoing basis
- **OpenTelemetry-native** — Built on the OpenTelemetry Collector, aligning with the industry-standard observability framework

Why this approach is recommended

OTel Container Insights is recommended over other Container Insights approaches because it provides the broadest signal coverage (metrics and logs) with the simplest deployment model. Other approaches are in maintenance-only mode or require more complex manual configuration.

The following table compares OTel Container Insights with Enhanced Container Insights (Classic).

Consideration	OTel Container Insights	Enhanced Container Insights (Classic)
Signals	Metrics, Logs	Metrics, Logs
Enhanced observability	Yes	Yes
Maintenance status	Active development	Maintenance
Deployment complexity	Low	Low

Metric sources and receivers

OTel Container Insights collects metrics from open-source receivers and automatically enriches them with OpenTelemetry semantic conventions. Unlike Enhanced Container Insights (Classic), which uses proprietary metric names, OTel Container Insights preserves the original metric names from each source. This makes them compatible with existing PromQL dashboards and community documentation.

OTel Container Insights collects open-source metrics from multiple receivers using the OpenTelemetry Protocol (OTLP) at 30-second granularity. Metrics use their original source names (for example, `container_cpu_usage_seconds_total` from cAdvisor) rather than CloudWatch-format names, and you can query them with PromQL.

The following table lists the available receivers and their prerequisites.

Receiver	What it collects	Prerequisites
cAdvisor	Container CPU, memory, network, disk/filesystem metrics	Built into kubelet — no additional setup
Prometheus Node Exporter	Node-level CPU, memory, disk, filesystem, network, system, VMStat, netstat/socket metrics	Included in the add-on
Kube State Metrics	Pod, node, Deployment, DaemonSet, StatefulSet, ReplicaSet, Job, CronJob, Service, Namespace, PV, PVC metrics	Included in the add-on
Kubernetes API Server	API server and etcd metrics	Available on control plane
NVIDIA DCGM	GPU utilization, memory, power/thermal, throttling, error/reliability, NVLink metrics	NVIDIA device plugin and container toolkit required
AWS Neuron Monitor	NeuronCore, NeuronDevice, and Neuron system metrics	Neuron driver and device plugin required
AWS Elastic Fabric Adapter	EFA networking metrics	EFA device plugin required
NVMe	NVMe SMART health metrics	No additional setup

Each metric is enriched with up to 150 labels from the following three sources:

1. **Telemetry source native labels** — From the original receiver (for example, pod, `namespace`, `container` from cAdvisor)
2. **OpenTelemetry resource attributes** — Following OTel semantic conventions for Kubernetes, Host, and Cloud
3. **Kubernetes pod and node labels** — All labels from the Kubernetes API, prefixed with `k8s.pod.label.*` and `k8s.node.label.*`

Dual publishing and migration

OTel Container Insights is disabled by default. To enable it, set `otelContainerInsights.enabled` to `true` in the add-on configuration.

Starting with add-on version v6.2.0, the add-on supports publishing metrics through both Enhanced Container Insights (Classic) and OTel Container Insights simultaneously. This allows you to validate the new metrics pipeline before fully migrating.

To enable OTel Container Insights alongside Enhanced Container Insights (Classic), run the following command.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"containerInsights":
{"enabled":true},"otelContainerInsights":{"enabled":true}}'
```

To disable OTel Container Insights and keep only Enhanced Container Insights (Classic), run the following command.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"otelContainerInsights":{"enabled":false}}'
```

To disable Enhanced Container Insights (Classic) and keep only OTel Container Insights, run the following command.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
```

```
  --configuration-values '{"containerInsights":
{"enabled":false},"otelContainerInsights":{"enabled":true}}'
```

Getting started

Use the following topics to set up and configure OTel Container Insights on your Amazon EKS clusters.

- Quick start: OTel Container Insights on Amazon EKS — Get started with OTel Container Insights using the fastest setup path.
- Enable OTel Container Insights from the console — Enable OTel Container Insights through the AWS Management Console.
- Deploy OTel Container Insights with Helm — Deploy OTel Container Insights using the Helm chart for flexible, GitOps-friendly management.
- Deploy OTel Container Insights with CloudFormation — Deploy OTel Container Insights using a CloudFormation template.
- Sending logs to Amazon CloudWatch — Configure log collection for your Amazon EKS clusters.
- Advanced configuration for OTel Container Insights on Amazon EKS — Customize and fine-tune OTel Container Insights for your environment.

Quick start: OTel Container Insights on Amazon EKS

This guide walks you through enabling OTel Container Insights on an existing Amazon EKS cluster. By the end of this procedure, your cluster sends infrastructure metrics and container logs to Amazon CloudWatch with Enhanced Observability enabled.

You can enable OTel Container Insights in two ways: by using the AWS Management Console (fastest) or by using the AWS CLI. Both approaches install the same `amazon-cloudwatch-observability` EKS add-on with the OTel Container Insights configuration. You don't need manual agent deployment, Helm charts, or custom collector pipelines. The entire process takes under 5 minutes.

Prerequisites

Before you enable OTel Container Insights, verify that you meet the following requirements.

- An existing Amazon EKS cluster running Kubernetes version 1.28 or later
- Platform version `eks.1` or later

- Version 6.2.0 or later of the `amazon-cloudwatch-observability` add-on
- AWS CLI version 2.15.0 or later (for CLI-based setup)
- `kubectl` configured to communicate with your target cluster
- IAM permissions: `eks:CreateAddon`, `eks:DescribeAddon`, and `iam:CreateServiceLinkedRole`
- The EKS Pod Identity Agent add-on installed on your cluster, or IAM Roles for Service Accounts (IRSA) configured
- Outbound internet access from the cluster to CloudWatch endpoints

Enable OTel Container Insights (console)

The AWS Management Console provides the fastest path to enable OTel Container Insights.

To enable OTel Container Insights by using the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/.
2. Choose **Clusters**, and then choose your cluster name.
3. Choose the **Observability** tab.
4. Choose **Enable Container Insights** and follow the on-screen instructions.

For a detailed console walkthrough, see Enable OTel Container Insights from the console.

Enable OTel Container Insights (AWS CLI)

Use the following steps to enable OTel Container Insights by using the AWS CLI.

Step 1: Create the IAM role

Create an IAM role that allows the CloudWatch Observability add-on to send data to CloudWatch.

To create the CloudWatch Observability add-on IAM role

1. Run the following command to create the role with a trust policy for EKS Pod Identity.

```
aws iam create-role \
  --role-name EKS-CloudWatch-Observability-Role \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
```

```
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }]
  }'
```

2. Attach the `CloudWatchAgentServerPolicy` managed policy to the role.

```
aws iam attach-role-policy \
  --role-name EKS-CloudWatch-Observability-Role \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Step 2: Create the Pod Identity association

Associate the IAM role with the CloudWatch agent service account in your cluster.

To create the Pod Identity association

- Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster and *`account-id`* with your AWS account ID.

```
aws eks create-pod-identity-association \
  --cluster-name cluster-name \
  --namespace amazon-cloudwatch \
  --service-account cloudwatch-agent \
  --role-arn arn:aws:iam::account-id:role/EKS-CloudWatch-Observability-Role
```

Step 3: Install the Amazon CloudWatch Observability add-on

Install the `amazon-cloudwatch-observability` add-on with OTel Container Insights enabled.

To install the add-on

- Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

```
aws eks create-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"otelContainerInsights":{"enabled":true}}'
```

> **Important**
>
> The `otelContainerInsights.enabled` configuration is required. OTel Container Insights is not enabled by default.

Step 4: Verify the add-on status

Confirm that the add-on installed successfully.

To verify the add-on status

- Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

  ```
  aws eks describe-addon \
    --cluster-name cluster-name \
    --addon-name amazon-cloudwatch-observability \
    --query "addon.status" \
    --output text
  ```

  The expected output is `ACTIVE`.

Step 5: Confirm agent pods are running

Verify that the CloudWatch agent pods are running in the `amazon-cloudwatch` namespace.

To confirm agent pods are running

- Run the following command.

  ```
  kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
  ```

  All agent pods must show `Running` status.

Verify data in CloudWatch

After you complete the setup, Container Insights data appears in CloudWatch within 3 to 5 minutes.

Check metrics

To check metrics in CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Query Studio**.
3. Search for metrics such as `container_cpu_usage_seconds_total` by using PromQL.

Check logs

To verify that log groups exist for your cluster, run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws logs describe-log-groups \
  --log-group-name-prefix "/aws/containerinsights/cluster-name" \
  --query "logGroups[].logGroupName" \
  --output table
```

Expected time-to-data

The following table shows the expected latency for each signal type after you enable OTel Container Insights.

Signal	Expected latency
Infrastructure metrics	2–3 minutes
Container logs	2–3 minutes
Performance log events	3–5 minutes

Troubleshooting

Use the following guidance to resolve common issues when you enable OTel Container Insights on Amazon EKS.

Add-on status shows CREATE_FAILED or DEGRADED

Symptom: When you run `aws eks describe-addon`, the status shows `CREATE_FAILED` or `DEGRADED`.

Cause: The add-on installation failed, typically because of insufficient IAM permissions or a missing Pod Identity association.

Solution: Complete the following steps to resolve this issue.

1. Run the following command to check for detailed error information. Replace *cluster-name* with the name of your cluster.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.health"
```

2. Verify that the IAM role exists and has the `CloudWatchAgentServerPolicy` attached.
3. Verify that the Pod Identity association targets the correct namespace (`amazon-cloudwatch`) and service account (`cloudwatch-agent`).
4. Delete the failed add-on and reinstall it after you resolve the issue.

```
aws eks delete-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability
```

Agent pods are in CrashLoopBackOff or Pending state

Symptom: When you run `kubectl get pods -n amazon-cloudwatch`, one or more pods show `CrashLoopBackOff` or `Pending` status.

Cause: The agent pods can't start because of insufficient node resources, missing permissions, or network connectivity issues.

Solution: Complete the following steps to resolve this issue.

1. Check the pod events for detailed error messages.

```
kubectl describe pod -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

2. Check the agent container logs for startup errors.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
tail=50
```

3. Verify that your nodes have sufficient CPU and memory available for the agent pods.
4. Verify that the EKS Pod Identity Agent add-on is installed and running.

```
kubectl get pods -n kube-system -l app.kubernetes.io/name=eks-pod-identity-agent
```

Metrics not appearing in CloudWatch after 5 minutes

Symptom: The agent pods show `Running` status, but no metrics appear in CloudWatch after 5 minutes.

Cause: The agent can't send data to CloudWatch, typically because of network restrictions or incorrect IAM permissions.

Solution: Complete the following steps to resolve this issue.

1. Verify that the agent pods can reach CloudWatch endpoints. Check that your VPC security groups and network ACLs allow outbound HTTPS traffic (port 443) to CloudWatch endpoints.
2. Check the agent logs for permission errors or connection timeouts.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
tail=100 | grep -i "error\|timeout\|denied"
```

3. Verify that the IAM role has the `CloudWatchAgentServerPolicy` policy attached and that the trust policy allows `pods.eks.amazonaws.com`.
4. If you use a VPC endpoint for CloudWatch, confirm that the endpoint policy allows the required actions.

Enable OTel Container Insights from the console

You can enable OTel Container Insights directly from the AWS Management Console with a minimal-click workflow. This approach is ideal if you prefer a UI-driven experience and want to enable full observability on an existing Amazon EKS cluster without writing infrastructure code or running CLI commands.

Prerequisites

Before you enable OTel Container Insights from the console, verify that you meet the following requirements.

- You are signed in to the Amazon EKS console.
- An existing Amazon EKS cluster running Kubernetes version 1.28 or later.
- Version 6.2.0 or later of the `amazon-cloudwatch-observability` add-on.
- IAM permissions: `eks:DescribeCluster`, `eks:UpdateClusterConfig`, `eks:ListAddons`, `eks:CreateAddon`, `eks:DescribeAddon`, `iam:AttachRolePolicy`, and `iam:CreateServiceLinkedRole`.
- The node IAM role must have the `CloudWatchAgentServerPolicy` managed policy attached.
- The cluster is in `ACTIVE` state.

Enable OTel Container Insights

Use the following procedure to enable OTel Container Insights from the Amazon EKS console.

To enable OTel Container Insights by using the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/.
2. Choose **Clusters**, and then choose your cluster name.
3. Choose the **Observability** tab.
4. Next to **Container Insights** status, choose **Manage**.
5. Select the **OTel Container Insights** configuration.
6. Confirm the IAM role for the add-on. The console auto-creates the role if one doesn't already exist.
7. Choose **Enable**.

What the console configures

When you enable OTel Container Insights from the console, the console performs the following actions on your behalf.

- Creates and configures the Pod Identity association for the `amazon-cloudwatch` namespace
- Installs the `amazon-cloudwatch-observability` Amazon EKS add-on with version 6.2.0 or later and `otelContainerInsights.enabled=true`
- Attaches the `CloudWatchAgentServerPolicy` to the designated IAM role

Verify that data appears in CloudWatch

After you enable OTel Container Insights, data appears in CloudWatch within 5 minutes.

To verify Container Insights data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**, and then choose **Container Insights**.
3. Choose your cluster from the cluster list.

The Container Insights dashboards populate with metrics data from your cluster.

Troubleshooting

Use the following guidance to resolve common issues when you enable OTel Container Insights from the console.

The Enable button is unavailable

Symptom: The **Enable** button is greyed out and you can't choose it.

Cause: The cluster might be in an updating state, or you don't have sufficient IAM permissions.

Solution: Complete the following steps to resolve this issue.

1. Verify that the cluster status is `ACTIVE` on the cluster details page.
2. Verify that your IAM user or role has the required permissions listed in the prerequisites.
3. If the cluster is updating, wait for the update to complete and try again.

Status shows DEGRADED after you enable

Symptom: After you enable OTel Container Insights, the add-on status shows `DEGRADED`.

Cause: The add-on installed but can't function correctly, typically because of IAM role permission issues or node connectivity problems.

Solution: Complete the following steps to resolve this issue.

1. Verify that the IAM role has the `CloudWatchAgentServerPolicy` managed policy attached.
2. Verify that the cluster nodes have outbound connectivity to CloudWatch endpoints.

3. Check the add-on health details in the Amazon EKS console by choosing the **Add-ons** tab on the cluster details page.

Deploy OTel Container Insights with Helm

The Amazon CloudWatch Observability Helm chart provides a flexible deployment option for OTel Container Insights. It installs the CloudWatch agent (OTel-based) with the OpenTelemetry filelog receiver for log collection. The Helm chart is maintained at https://github.com/aws-observability/helm-charts.

Use the Helm chart when you need one or more of the following capabilities:

- Fine-grained control over agent configuration
- OTel-native log collection using the filelog receiver (no Fluent Bit dependency)
- Custom TLS certificate management through cert-manager
- Non-EKS Kubernetes clusters (ROSA, self-managed Kubernetes)
- GitOps workflows with ArgoCD or Flux

Prerequisites

Before you deploy OTel Container Insights with Helm, verify that you meet the following requirements.

- An existing Amazon EKS cluster running Kubernetes version 1.28 or later
- Helm version 3.9 or later
- `kubectl` configured to communicate with your target cluster
- IAM permissions: `CloudWatchAgentServerPolicy` managed policy attached to the agent role
- An OpenID Connect (OIDC) provider configured for IAM Roles for Service Accounts (IRSA)
- Outbound internet access from the cluster to CloudWatch endpoints

Install the Helm chart

Complete the following steps to deploy OTel Container Insights by using the Helm chart.

Step 1: Add the Helm repository

Add the AWS Observability Helm chart repository to your local Helm configuration.

To add the Helm repository

1. Run the following command to add the repository.

```
helm repo add aws-observability \
  https://aws-observability.github.io/helm-charts
```

2. Update the repository to get the latest chart versions.

```
helm repo update
```

Step 2: Create the IAM role

Create an IAM role that allows the CloudWatch agent to send data to CloudWatch. This role uses IRSA to associate IAM permissions with a Kubernetes service account.

To create the IAM role for the CloudWatch agent

1. Retrieve the OIDC issuer URL for your cluster. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks describe-cluster \
  --name cluster-name \
  --query "cluster.identity.oidc.issuer" \
  --output text
```

2. Create the IAM role with a trust policy for IRSA. Replace the example account ID (111122223333) with your AWS account ID, and replace *oidc-id* with the OIDC provider ID from the preceding step (the portion after `https://oidc.eks.region.amazonaws.com/id/`).

```
aws iam create-role \
  --role-name EKS-CW-Observability-Role \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/oidc-id"
      },
```

```
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/oidc-id:sub":
 "system:serviceaccount:amazon-cloudwatch:cloudwatch-agent"
        }
      }
    }]
  }'
```

3. Attach the `CloudWatchAgentServerPolicy` managed policy to the role.

```
aws iam attach-role-policy \
  --role-name EKS-CW-Observability-Role \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Step 3: Install the chart

Install the Amazon CloudWatch Observability Helm chart with OTel Container Insights enabled.

To install the Helm chart

- Run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster, the example AWS Region (`us-east-1`) with your own Region, and the example account ID (111122223333) with your AWS account ID.

```
helm install amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  --namespace amazon-cloudwatch \
  --create-namespace \
  --set clusterName=cluster-name \
  --set region=us-east-1 \
  --set agent.serviceAccount.name=cloudwatch-agent \
  --set "agent.serviceAccount.annotations.eks\\.amazonaws\\.com/role-
arn=arn:aws:iam::111122223333:role/EKS-CW-Observability-Role" \
  --set otelContainerInsights.enabled=true
```

> **Important**
>
> The `otelContainerInsights.enabled` parameter is required. OTel Container Insights is not enabled by default.

Step 4: Verify the installation

Confirm that the Helm release and agent pods deployed successfully.

To verify the Helm installation

1. Verify that the Helm release status shows `deployed`.

```
helm list -n amazon-cloudwatch
```

2. Confirm that the operator pod is running.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=amazon-cloudwatch-observability
```

3. Confirm that the CloudWatch agent pods are running on all nodes.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

   All agent pods must show `Running` status.

Key configuration options

The following tables describe the key Helm chart values that you can configure. Pass these values by using the `--set` flag or a custom `values.yaml` file.

Cluster settings

Parameter	Default	Description
`clusterName`	—	The name of the Amazon EKS cluster. Required.

Parameter	Default	Description
region	—	The AWS Region where the cluster runs. Required.
otelContainerInsights.enabled	false	Enables OTel Container Insights with the filelog receiver.

Container logs (OTel filelog receiver)

Parameter	Default	Description
containerLogs.enabled	true	Enables container log collection by using the OTel filelog receiver.
containerLogs.logGroupName	/aws/containerinsights/ *clustername* /application	The CloudWatch Logs log group name for container logs.
containerLogs.logRetentionDays	7	The number of days to retain container logs in CloudWatch Logs.

Agent (metrics)

Parameter	Default	Description
agent.enabled	true	Enables the CloudWatch agent DaemonSet for metric collection.
agent.serviceAccount.name	cloudwatch-agent	The name of the Kubernetes service account for the agent.

Parameter	Default	Description
`agent.resources.requests.cpu`	`100m`	The CPU request for the agent container.
`agent.resources.requests.memory`	`128Mi`	The memory request for the agent container.
`agent.resources.limits.cpu`	`200m`	The CPU limit for the agent container.
`agent.resources.limits.memory`	`256Mi`	The memory limit for the agent container.

GPU monitoring

Parameter	Default	Description
`agent.config.logs.metrics_collected.kubernetes.enhanced_container_insights`	`true`	Enables Enhanced Container Insights metrics, including GPU metrics.
`dcgmExporter.enabled`	`false`	Enables the DCGM exporter for NVIDIA GPU metrics. Requires NVIDIA GPU nodes.
`neuronMonitor.enabled`	`false`	Enables the Neuron monitor for AWS Inferentia and Trainium metrics.

TLS certificate management

The CloudWatch Observability operator requires TLS certificates for webhook communication. By default, the chart generates self-signed certificates. You can use cert-manager to manage certificates automatically.

Self-signed certificates (default)

By default, the Helm chart generates a self-signed CA and issues certificates from that CA. No additional configuration is required. The chart automatically rotates these certificates during upgrades.

cert-manager integration

You can use cert-manager to automate certificate issuance and renewal. This approach is useful when your organization requires externally signed certificates or centralized certificate management.

To configure cert-manager integration

1. Verify that cert-manager is installed in your cluster.

```
kubectl get pods -n cert-manager
```

2. Install the Helm chart with cert-manager enabled. Replace *cluster-name* with the name of your Amazon EKS cluster, the example AWS Region (us-east-1) with your own Region, and the example account ID (111122223333) with your AWS account ID.

```
helm install amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  --namespace amazon-cloudwatch \
  --create-namespace \
  --set clusterName=cluster-name \
  --set region=us-east-1 \
  --set agent.serviceAccount.name=cloudwatch-agent \
  --set "agent.serviceAccount.annotations.eks\\.amazonaws\\.com/role-
arn=arn:aws:iam::111122223333:role/EKS-CW-Observability-Role" \
  --set otelContainerInsights.enabled=true \
  --set admissionWebhooks.certManager.enabled=true
```

When cert-manager is enabled, the chart creates a `Certificate` resource that cert-manager uses to issue and manage the webhook TLS certificate automatically.

> **Note**
>
> You can also use cert-manager with external issuers such as HashiCorp Vault. To configure an external issuer, set `admissionWebhooks.certManager.issuerRef.name` and

`admissionWebhooks.certManager.issuerRef.kind` to match your cert-manager issuer.

Upgrade the chart

Upgrade the Helm chart to apply new configuration values or to update to a newer chart version.

To upgrade the Helm chart

1. Update the Helm repository to get the latest chart versions.

```
helm repo update
```

2. Run the upgrade command with your desired values. Replace *`cluster-name`* with the name of your Amazon EKS cluster, the example AWS Region (`us-east-1`) with your own Region, and the example account ID (111122223333) with your AWS account ID.

```
helm upgrade amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  --namespace amazon-cloudwatch \
  --set clusterName=cluster-name \
  --set region=us-east-1 \
  --set agent.serviceAccount.name=cloudwatch-agent \
  --set "agent.serviceAccount.annotations.eks\\.amazonaws\\.com/role-
arn=arn:aws:iam::111122223333:role/EKS-CW-Observability-Role" \
  --set otelContainerInsights.enabled=true
```

3. Verify that the upgrade completed successfully.

```
helm list -n amazon-cloudwatch
```

The revision number must increment and the status must show `deployed`.

Tip

To avoid losing configuration values during upgrades, store your custom values in a `values.yaml` file and pass it with the `-f values.yaml` flag.

Uninstall the chart

To remove OTel Container Insights from your cluster, uninstall the Helm release.

To uninstall the Helm chart

1. Run the following command to uninstall the release.

```
helm uninstall amazon-cloudwatch-observability \
  --namespace amazon-cloudwatch
```

2. Optionally, delete the namespace if it is no longer needed.

```
kubectl delete namespace amazon-cloudwatch
```

After you uninstall the Helm chart, the IAM role and OIDC provider configuration remain in your AWS account. Delete these resources separately if they are no longer needed.

```
aws iam detach-role-policy \
  --role-name EKS-CW-Observability-Role \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

aws iam delete-role \
  --role-name EKS-CW-Observability-Role
```

Verify data in CloudWatch

After you install the Helm chart, Container Insights data appears in CloudWatch within 3 to 5 minutes.

Check metrics

To check metrics in CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Container Insights**.
3. Verify that your cluster appears in the cluster list and that infrastructure metrics are populating.

Check logs

To verify that log groups exist for your cluster, run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws logs describe-log-groups \
  --log-group-name-prefix "/aws/containerinsights/cluster-name" \
  --query "logGroups[].logGroupName" \
  --output table
```

Troubleshooting

Use the following guidance to resolve common issues when you deploy OTel Container Insights with Helm.

Operator pod in CrashLoopBackOff

Symptom: When you run `kubectl get pods -n amazon-cloudwatch`, the operator pod shows `CrashLoopBackOff` status.

Cause: The operator pod can't start because of TLS certificate issues or insufficient permissions.

Solution: Complete the following steps to resolve this issue.

1. Check the operator pod logs for certificate-related errors.

   ```
   kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=amazon-cloudwatch-
   observability --tail=50
   ```

2. Verify that the webhook TLS secret exists in the namespace.

   ```
   kubectl get secrets -n amazon-cloudwatch | grep webhook
   ```

3. If you use cert-manager, verify that the `Certificate` resource status shows `Ready`.

   ```
   kubectl get certificate -n amazon-cloudwatch
   ```

4. If the certificate is missing or invalid, uninstall and reinstall the chart to regenerate certificates.

   ```
   helm uninstall amazon-cloudwatch-observability -n amazon-cloudwatch
   ```

Agent pods not scheduled on all nodes

Symptom: The agent DaemonSet shows fewer pods than the number of nodes in your cluster.

Cause: Node taints, resource constraints, or node selectors prevent the agent pods from scheduling on certain nodes.

Solution: Complete the following steps to resolve this issue.

1. Check the DaemonSet status for scheduling issues.

```
kubectl get daemonset -n amazon-cloudwatch cloudwatch-agent
```

2. Check for unschedulable pods and view their events.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
field-selector=status.phase!=Running
```

3. If nodes have taints, add tolerations to the Helm chart values. For example, to tolerate all taints, upgrade the chart with the following flag.

```
helm upgrade amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  --namespace amazon-cloudwatch \
  --reuse-values \
  --set "agent.tolerations[0].operator=Exists"
```

4. Verify that the agent pods now run on all nodes.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent -o
 wide
```

Deploy OTel Container Insights with CloudFormation

You can deploy OTel Container Insights as infrastructure as code by using a CloudFormation template. This approach is best for teams that manage clusters through IaC pipelines and want repeatable, version-controlled deployments.

The template installs the `amazon-cloudwatch-observability` EKS add-on with OTel Container Insights enabled. If you prefer Kubernetes-native package management, see Deploy OTel Container Insights with Helm.

Prerequisites

Before you deploy OTel Container Insights with CloudFormation, verify that you meet the following requirements.

- An existing Amazon EKS cluster running Kubernetes version 1.28 or later
- AWS CLI version 2.15.0 or later
- `kubectl` configured to communicate with your target cluster
- IAM permissions for CloudFormation stack creation
- Outbound internet access from the cluster to CloudWatch endpoints

Deploy with CloudFormation

Use a CloudFormation template to deploy the CloudWatch Observability add-on, the required IAM role, and the Pod Identity association in a single stack. This template uses the `AWS::EKS::Addon` resource to install the add-on with OTel Container Insights enabled.

CloudFormation template

The following template creates the complete set of resources that you need to enable OTel Container Insights on your Amazon EKS cluster.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy OTel Container Insights on an EKS cluster

Parameters:
  ClusterName:
    Type: String
    Description: The name of your EKS cluster

Resources:
  CloudWatchAgentRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: EKS-CloudWatch-Observability-Role
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
```

```
            Service: pods.eks.amazonaws.com
          Action:
            - sts:AssumeRole
            - sts:TagSession
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

PodIdentityAssociation:
  Type: AWS::EKS::PodIdentityAssociation
  Properties:
    ClusterName: !Ref ClusterName
    Namespace: amazon-cloudwatch
    ServiceAccount: cloudwatch-agent
    RoleArn: !GetAtt CloudWatchAgentRole.Arn

CloudWatchObservabilityAddon:
  Type: AWS::EKS::Addon
  DependsOn: PodIdentityAssociation
  Properties:
    ClusterName: !Ref ClusterName
    AddonName: amazon-cloudwatch-observability
    AddonVersion: v6.2.0-eksbuild.1
    ConfigurationValues: '{"otelContainerInsights":{"enabled":true}}'
    ServiceAccountRoleArn: !GetAtt CloudWatchAgentRole.Arn
    ResolveConflicts: OVERWRITE
```

This template creates the following resources:

- An IAM role with the `CloudWatchAgentServerPolicy` managed policy attached
- An EKS Pod Identity association that maps the role to the CloudWatch agent service account
- The `amazon-cloudwatch-observability` EKS add-on with OTel Container Insights enabled

To deploy the CloudFormation stack

Use the AWS CLI to create the CloudFormation stack from the template.

To deploy the stack

1. Save the preceding template to a file named `otel-container-insights.yaml`.
2. Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

```
aws cloudformation create-stack \
  --stack-name otel-container-insights \
  --template-body file://otel-container-insights.yaml \
  --parameters ParameterKey=ClusterName,ParameterValue=cluster-name \
  --capabilities CAPABILITY_NAMED_IAM
```

3. Wait for the stack creation to complete.

```
aws cloudformation wait stack-create-complete \
  --stack-name otel-container-insights
```

4. Verify that the stack status is `CREATE_COMPLETE`.

```
aws cloudformation describe-stacks \
  --stack-name otel-container-insights \
  --query "Stacks[0].StackStatus" \
  --output text
```

Verify the deployment

After the CloudFormation stack creation completes, verify that the add-on is running and sending data to CloudWatch.

To verify the deployment

1. Confirm that the CloudWatch agent pods are running.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

   All pods must show `Running` status.

2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the navigation pane, choose **Container Insights**.
4. Verify that your cluster appears in the cluster list and that metrics are populating.

Metrics typically appear in CloudWatch within 3 to 5 minutes after the deployment completes.

Remove the deployment

To remove all resources that the CloudFormation template created, delete the stack.

```
aws cloudformation delete-stack \
  --stack-name otel-container-insights
```

Sending logs to Amazon CloudWatch

OTel Container Insights collects and sends container logs to Amazon CloudWatch Logs by using the OpenTelemetry Collector's log pipeline. The Amazon CloudWatch Observability EKS add-on deploys an OpenTelemetry Collector as a DaemonSet that uses the filelog receiver to tail container log files, enrich them with Kubernetes metadata, and export them to CloudWatch Logs through the CloudWatch Logs exporter.

No additional setup is required for basic log collection. Log collection is enabled by default when you follow the Quick start: OTel Container Insights on Amazon EKS.

Log collection costs

Enabling log collection ingests all container stdout and stderr from every pod in the cluster, which can significantly increase CloudWatch Logs costs. To control cost, filter logs before ingestion and set a retention policy on the log groups. For information about filtering, see Log filtering. For information about retention policies, see Set log retention.

Correlating logs with traces

Collecting container logs does not by itself correlate them with traces in CloudWatch Application Signals. To correlate traces with logs, instrument your application for Application Signals by using the enablement guide for your architecture, such as Enable your applications on Amazon EKS clusters, and then configure your logging output as described in Enable trace to log correlation.

Prerequisites

Before you configure log collection, verify that you meet the following requirements.

- OTel Container Insights installed and the `amazon-cloudwatch-observability` add-on active on your cluster
- IAM permissions: `logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents`, `logs:DescribeLogGroups`, and `logs:DescribeLogStreams` (included in the `CloudWatchAgentServerPolicy` managed policy)
- Node access: the collector requires access to `/var/log/pods` on each node

Log groups and sources

The following table describes the log group that OTel Container Insights creates and the source it collects from.

Log group	Source	Contents
`/aws/containerinsights/` *`cluster-name`* `/application`	`/var/log/pods/**/*.log`	All container stdout and stderr logs

> **Note**
>
> Host and dataplane logs will be available in a future release.

How the OTel log pipeline works

The add-on's OpenTelemetry Collector runs a log pipeline with the following components.

- **Receivers** – The `filelog` receiver tails container log files from `/var/log/pods/`.
- **Processors** – The `k8sattributes` processor enriches logs with Kubernetes metadata. The `batch` processor batches log records before export. The `resource` processor appends resource attributes.
- **Exporters** – The `awscloudwatchlogs` exporter sends log records to CloudWatch Logs.

Log enrichment

The pipeline enriches every log record with the following attributes.

- **Kubernetes resource attributes** – `k8s.pod.name`, `k8s.namespace.name`, `k8s.container.name`, `k8s.node.name`, and `k8s.deployment.name`
- **Pod labels** – All pod labels as `k8s.pod.label.*` attributes
- **Cloud attributes** – `cloud.region`, `cloud.account.id`, and `cloud.platform`
- **Cluster attributes** – `k8s.cluster.name`

Customizing log collection

You can customize log collection by updating the add-on configuration values. The following sections describe common customization options.

Disable log collection

To disable log collection entirely, update the add-on configuration with container logs disabled.

To disable log collection

- Run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"containerLogs":{"enabled":false}}' \
  --resolve-conflicts OVERWRITE
```

Set log retention

By default, CloudWatch Logs retains log data indefinitely. To control storage costs, you can set a retention policy on the log group.

To set log retention

- Run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster. Replace *30* with the number of days to retain logs.

```
aws logs put-retention-policy \
  --log-group-name "/aws/containerinsights/cluster-name/application" \
  --retention-in-days 30
```

Verification

To verify that log collection is working, check that the expected log groups exist and contain data.

To verify log collection

- Run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

  ```
  aws logs describe-log-groups \
    --log-group-name-prefix "/aws/containerinsights/cluster-name" \
    --query "logGroups[].{Name:logGroupName,StoredBytes:storedBytes}" \
    --output table
  ```

  The output displays the log group names and the number of stored bytes. A non-zero value for `StoredBytes` confirms that the pipeline is delivering logs.

Troubleshooting

Use the following guidance to resolve common log collection issues.

Log groups not created after 5 minutes

Symptom: The `/aws/containerinsights/cluster-name/application` log group does not appear in CloudWatch Logs after 5 minutes.

Cause: The collector does not have the required IAM permissions to create log groups and log streams.

Solution: Verify that the IAM role associated with the collector has the `CloudWatchAgentServerPolicy` managed policy attached. This policy includes the `logs:CreateLogGroup` and `logs:CreateLogStream` permissions.

Application log group exists but is empty

Symptom: The application log group exists in CloudWatch Logs, but it contains no log streams or log events.

Cause: This issue occurs when containers are not writing to stdout or stderr, or the filelog receiver cannot access `/var/log/pods/` on the node.

Solution: Complete the following steps to resolve this issue.

1. Verify that your application containers write logs to stdout or stderr.
2. Check that the collector DaemonSet pod has a volume mount for `/var/log/pods`.

```
kubectl get daemonset -n amazon-cloudwatch -o yaml | grep -A 5 "var/log/pods"
```

3. Check the collector logs for file access errors.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
tail=50 | grep -i "error\|permission"
```

High CloudWatch Logs costs

Symptom: CloudWatch Logs ingestion or storage costs are higher than expected.

Cause: High-volume namespaces (such as load testing or monitoring namespaces) generate large volumes of logs.

Solution: Set a retention policy on the log group to automatically delete older logs. For instructions, see Set log retention.

Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart

You can use either the Amazon CloudWatch Observability EKS add-on or the Amazon CloudWatch Observability Helm chart to install the CloudWatch Agent and the Fluent-bit agent on an Amazon EKS cluster. You can also use the Helm chart to install the CloudWatch Agent and the Fluent-bit agent on a Kubernetes cluster that is not hosted on Amazon EKS.

Using either method on an Amazon EKS cluster enables both Container Insights for Amazon EKS and CloudWatch Application Signals by default. Both features help you to collect infrastructure metrics, application performance telemetry, and container logs from the cluster.

From version `v6.2.0-eksbuild.1` or later of the add-on, Container Insights with OpenTelemetry metrics can be enabled, which collects metrics using the OpenTelemetry Protocol (OTLP) and supports PromQL queries. For more information, see OTel Container Insights (Recommended).

With Container Insights with OpenTelemetry for Amazon EKS, Container Insights metrics are charged per GB ingested or per observation if you are using Enhanced Container Insights (Classic). For Application Signals, billing is based on inbound requests to your applications, outbound

requests from your applications, and each configured service level objective (SLO). Each inbound request received generates one application signal, and each outbound request made generates one application signal. Every SLO creates two application signals per measurement period. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Both methods enable Container Insights on both Linux and Windows worker nodes in the Amazon EKS cluster. To enable Container Insights on Windows, you must use version 1.5.0 or later of the Amazon EKS add-on or the Helm chart. Application Signals is not supported on Windows in Amazon EKS clusters.

The Amazon CloudWatch Observability EKS add-on is supported on Amazon EKS clusters running with Kubernetes version 1.23 or later.

When you install the add-on or the Helm chart, you must also grant IAM permissions to enable the CloudWatch agent to send metrics, logs, and traces to CloudWatch. There are two ways to do this:

- Attach a policy to the IAM role of your worker nodes. This option grants permissions to worker nodes to send telemetry to CloudWatch.
- Use an IAM role for service accounts for the agent pods, and attach the policy to this role. This works only for Amazon EKS clusters. This option gives CloudWatch access only to the appropriate agent pods.

Topics

- Option 1: Install using EKS Pod Identity
- Option 2: Install with IAM permissions on worker nodes
- Option 3: Install using IAM service account role (applies only to using the add-on)
- (Optional) Additional configuration
- Collect Java Management Extensions (JMX) metrics
- Enable Kueue metrics
- Appending OpenTelemetry collector configuration files
- Enabling APM through Application Signals for your Amazon EKS cluster
- Considerations for large Kubernetes clusters
- Considerations for Amazon EKS Hybrid Nodes
- Troubleshooting the Amazon CloudWatch Observability EKS add-on or the Helm chart
- Opt out of Application Signals

Option 1: Install using EKS Pod Identity

If you use version 3.1.0 or later of the add-on, you can use EKS Pod Identity to grant the required permissions to the add-on. EKS Pod Identity is the recommended option and provides benefits such as least privilege, credential rotation, and auditability. Additionally, using EKS Pod Identity allows you to install the EKS add-on as part of the cluster creation itself.

To use this method, first follow the EKS Pod Identity association steps to create the IAM role and set up the EKS Pod Identity agent.

Then install the Amazon CloudWatch Observability EKS add-on. To install the add-on, you can use the AWS CLI, the Amazon EKS console, CloudFormation, or Terraform.

AWS CLI

To use the AWS CLI to install the Amazon CloudWatch Observability EKS add-on

Enter the following commands. Replace *my-cluster-name* with the name of your cluster and replace *111122223333* with your account ID. Replace *my-role* with the IAM role that you created in the EKS Pod Identity association step.

```
aws iam attach-role-policy \
--role-name my-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

aws eks create-addon \
--addon-name amazon-cloudwatch-observability \
--cluster-name my-cluster-name \
--pod-identity-associations serviceAccount=cloudwatch-
agent,roleArn=arn:aws:iam::111122223333:role/my-role
```

Amazon EKS console

To use the Amazon EKS console to add the Amazon CloudWatch Observability EKS add-on

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.

5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
   a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.
   b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:
   a. Select the **Version** you'd like to use.
   b. For **Add-on access**, select **EKS Pod Identity**
   c. If you don't have an IAM role configured, choose **Create recommended role**, then choose **Next** until you are at **Step 3 Name, review, and create**. You can change your role name if desired, otherwise, choose **Create Role**, and then return to the Add-on page and select the IAM role that you just created.
   d. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.
   e. Choose **Next**.
8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

CloudFormation

To use CloudFormation to install the Amazon CloudWatch Observability EKS add-on

1. First, run the following AWS CLI command to attach the necessary IAM policy to your IAM role. Replace *my-role* with the role that you created in the EKS Pod Identity association step.

```
aws iam attach-role-policy \
--role-name my-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

2. Then create the following resource. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-role* with the IAM role created in EKS Pod Identity association step. For more information, see AWS::EKS::Addon.

```
{
"Resources": {
    "EKSAddOn": {
        "Type": "AWS::EKS::Addon",
        "Properties": {
            "AddonName": "amazon-cloudwatch-observability",
            "ClusterName": "my-cluster-name",
            "PodIdentityAssociations": [
                {
                    "ServiceAccount": "cloudwatch-agent",
                    "RoleArn": "arn:aws:iam::111122223333:role/my-role"
                }
            ]
        }
    }
}
}
```

Terraform

To use Terraform to install the Amazon CloudWatch Observability EKS add-on

1. Use the following. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-service-account-role* with the IAM role created in EKS Pod Identity association step.

   For more information, see Resource: aws_eks_addon in the Terraform documentation.

2.
```
resource "aws_iam_role_policy_attachment" "CloudWatchAgentServerPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
  role       = "my-role"
}

resource "aws_eks_addon" "example" {
  cluster_name = "my-cluster-name"
  addon_name   = "amazon-cloudwatch-observability"
```

```
  pod_identity_associations {
  roleArn = "arn:aws:iam::111122223333:role/my-role"
  serviceAccount = "cloudwatch-agent"
  }
}
```

Option 2: Install with IAM permissions on worker nodes

To use this method, first attach the **CloudWatchAgentServerPolicy** IAM policy to your worker nodes by entering the following command. In this command, replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Then install the Amazon CloudWatch Observability EKS add-on. To install the add-on, you can use the AWS CLI, the console, CloudFormation, or Terraform.

AWS CLI

To use the AWS CLI to install the Amazon CloudWatch Observability EKS add-on

Enter the following command. Replace *my-cluster-name* with the name of your cluster.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability --cluster-name my-cluster-name
```

Amazon EKS console

To use the Amazon EKS console to add the Amazon CloudWatch Observability EKS add-on

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.

6. On the **Select add-ons** page, do the following:

   a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.

   b. Choose **Next**.

7. On the **Configure selected add-ons settings** page, do the following:

   a. Select the **Version** you'd like to use.

   b. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.

   c. Choose **Next**.

8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

CloudFormation

To use CloudFormation to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see AWS::EKS::Addon.

```
{
"Resources": {
    "EKSAddOn": {
        "Type": "AWS::EKS::Addon",
        "Properties": {
            "AddonName": "amazon-cloudwatch-observability",
            "ClusterName": "my-cluster-name"
        }
    }
}
}
```

Helm chart

To use the `amazon-cloudwatch-observability` Helm chart

1. You must have Helm installed to use this chart. For more information about installing Helm, see the Helm documentation.
2. After you have installed Helm, enter the following commands. Replace *my-cluster-name* with the name of your cluster, and replace *my-cluster-region* with the Region that the cluster runs in.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm repo update aws-observability
helm install --wait --create-namespace --namespace amazon-cloudwatch amazon-
cloudwatch-observability aws-observability/amazon-cloudwatch-observability --set
 clusterName=my-cluster-name --set region=my-cluster-region
```

Terraform

To use Terraform to install the Amazon CloudWatch Observability EKS add-on

Replace *my-cluster-name* with the name of your cluster. For more information, see Resource: aws_eks_addon.

```
resource "aws_eks_addon" "example" {
  addon_name   = "amazon-cloudwatch-observability"
  cluster_name = "my-cluster-name"
}
```

Option 3: Install using IAM service account role (applies only to using the add-on)

This method is valid only if you are using the Amazon CloudWatch Observability EKS add-on. Before using this method, verify the following prerequisites:

- You have a functional Amazon EKS cluster with nodes attached in one of the AWS Regions that supports Container Insights. For the list of supported Regions, see Container Insights.
- You have `kubectl` installed and configured for the cluster. For more information, see Installing `kubectl` in the *Amazon EKS User Guide*.

- You have `eksctl` installed. For more information, see Installing or updating `eksctl` in the *Amazon EKS User Guide*.

AWS CLI

To use the AWS CLI to install the Amazon CloudWatch Observability EKS add-on using the IAM service account role

1. Enter the following command to create an OpenID Connect (OIDC) provider, if the cluster doesn't have one already. For more information, see Configuring a Kubernetes service account to assume an IAM role in the *Amazon EKS User Guide*.

```
eksctl utils associate-iam-oidc-provider --cluster my-cluster-name --approve
```

2. Enter the following command to create the IAM role with the **CloudWatchAgentServerPolicy** policy attached, and configure the agent service account to assume that role using OIDC. Replace *my-cluster-name* with the name of your cluster, and replace *my-service-account-role* with the name of the role that you want to associate the service account with. If the role doesn't already exist, `eksctl` creates it for you.

```
eksctl create iamserviceaccount \
  --name cloudwatch-agent \
  --namespace amazon-cloudwatch --cluster my-cluster-name \
  --role-name my-service-account-role \
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --role-only \
  --approve
```

3. Install the add-on by entering the following command. Replace *my-cluster-name* with the name of your cluster, replace *111122223333* with your account ID, and replace *my-service-account-role* with the IAM role created in the previous step.

```
aws eks create-addon --addon-name amazon-cloudwatch-observability
 --cluster-name my-cluster-name --service-account-role-arn
 arn:aws:iam::111122223333:role/my-service-account-role
```

Amazon EKS console

To use the console to install the Amazon CloudWatch Observability EKS add-on using the IAM service account role

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. In the left navigation pane, choose **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon CloudWatch Observability EKS add-on for.
4. Choose the **Add-ons** tab.
5. Choose **Get more add-ons**.
6. On the **Select add-ons** page, do the following:
   a. In the **Amazon EKS-addons** section, select the **Amazon CloudWatch Observability** check box.
   b. Choose **Next**.
7. On the **Configure selected add-ons settings** page, do the following:
   a. Select the **Version** you'd like to use.
   b. For **Add-on access**, select **IAM roles for service accounts (IRSA)**
   c. Select the IAM role in the **Add-on access** box.
   d. (Optional) You can expand the **Optional configuration settings**. If you select **Override** for the **Conflict resolution method**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage.
   e. Choose **Next**.
8. On the **Review and add** page, choose **Create**. After the add-on installation is complete, you see your installed add-on.

(Optional) Additional configuration

Topics

- Opt out of collecting container logs
- Use a custom Fluent Bit configuration
- Manage Kubernetes tolerations for the installed pod workloads
- Opt out of accelerated compute metrics collection
- Use a custom CloudWatch agent configuration
- Manage admission webhook TLS certificates
- Collect Amazon EBS volume IDs

Opt out of collecting container logs

By default, the add-on uses Fluent Bit to collect container logs from all pods and then sends the logs to CloudWatch Logs. For information about which logs are collected, see Send logs to CloudWatch Logs.

> **Note**
>
> Neither the add-on or the Helm chart manage existing Fluentd or Fluent Bit resources in a cluster. You can delete the existing Fluentd or Fluent Bit resources before installing the add-on or Helm chart. Alternatively, to keep your existing setup and avoid having the add-on or the Helm chart from also installing Fluent Bit, you can disable it by following the instructions in this section.

To opt out of the collection of container logs if you are using the Amazon CloudWatch Observability EKS add-on, pass the following option when you create or update the add-on:

```
--configuration-values '{ "containerLogs": { "enabled": false } }'
```

To opt out of the collection of container logs if you are using the Helm chart, pass the following option when you create or update the add-on:

```
--set containerLogs.enabled=false
```

Use a custom Fluent Bit configuration

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, you can modify the Fluent Bit configuration when you create or update the add-on or Helm chart. You supply

the custom Fluent Bit configuration in the `containerLogs` root level section of the advanced configuration of the add-on or the value overrides in the Helm chart. Within this section, you supply the custom Fluent Bit configuration in the `config` section (for Linux) or `configWindows` section (for Windows). The `config` is further broken down into the following sub-sections:

- `service`– This section represents the `SERVICE` config to define the global behavior of the Fluent Bit engine.
- `customParsers`– This section represents any global `PARSER`s that you want to include that are capable of taking unstructured log entries and giving them a structure to make it easier for processing and further filtering.
- `extraFiles`– This section can be used to provide additional Fluent Bit `conf` files to be included. By default, the following 3 `conf` files are included:.
  - `application-log.conf`– A `conf` file for sending application logs from your cluster to the log group `/aws/containerinsights/`*`my-cluster-name`*`/application` in CloudWatch Logs.
  - `dataplane-log.conf`– A `conf` file for sending logs corresponding to your cluster's data plane components including the CRI logs, kubelet logs, kube-proxy logs and Amazon VPC CNI logs to the log group `/aws/containerinsights/`*`my-cluster-name`*`/dataplane` in CloudWatch Logs.
  - `host-log.conf`– A `conf` for sending logs from `/var/log/dmesg`, `/var/log/messages`, and `/var/log/secure` on Linux, and System `winlogs` on Windows, to the log group `/aws/containerinsights/`*`my-cluster-name`*`/host` in CloudWatch.

> **Note**
>
> Provide the full configuration for each of these individual sections even if you are modifying only one field within a sub-section. We recommend that you use the default configuration provided below as a baseline and then modify it accordingly so that you don't disable functionality that is enabled by default. You can use the following YAML configuration when modifying the advanced config for the Amazon EKS add-on or when you supply value overrides for the Helm chart.

To find the `config` section for your cluster, see aws-observability / helm-charts on GitHub and find the release corresponding to the version of the add-on or Helm chart that you are installing.

Then navigate to `/charts/amazon-cloudwatch-observability/values.yaml` to find the `config` section (for Linux) and `configWindows` section (for Windows) within the `fluentBit` section under `containerLogs`.

As an example, the default Fluent Bit configuration for version 1.7.0 can be found here.

We recommend that you provide the `config` as YAML when you supply it using the Amazon EKS add-on's advanced config or when you supply it as value overrides for your Helm installation. Be sure that the YAML conforms to the following structure.

```
containerLogs:
  fluentBit:
config:
  service: |
    ...
  customParsers: |
    ...
  extraFiles:
    application-log.conf: |
      ...
    dataplane-log.conf: |
      ...
    host-log.conf: |
      ...
```

The following example `config` changes the global setting for the flush interval to be 45 seconds. Even though the only modification is to the `Flush` field, you must still provide the full `SERVICE` definition for the service sub-section. Because this example didn't specify overrides for the other sub-sections, the defaults are used for them.

```
containerLogs:
  fluentBit:
config:
  service: |
    [SERVICE]
      Flush                     45
      Grace                     30
      Log_Level                 error
      Daemon                    off
      Parsers_File              parsers.conf
      storage.path              /var/fluent-bit/state/flb-storage/
      storage.sync              normal
```

```
      storage.checksum          off
      storage.backlog.mem_limit 5M
```

The following example configuration includes an extra Fluent bit `conf` file. In this example, you are adding a custom `my-service.conf` under `extraFiles` and it will be included in addition to the three default `extraFiles`.

```
containerLogs:
  fluentBit:
config:
  extraFiles:
    my-service.conf: |
      [INPUT]
        Name              tail
        Tag               myservice.*
        Path              /var/log/containers/*myservice*.log
        DB                /var/fluent-bit/state/flb_myservice.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Ignore_Older      1d
        Refresh_Interval  10

      [OUTPUT]
        Name                cloudwatch_logs
        Match               myservice.*
        region              ${AWS_REGION}
        log_group_name      /aws/containerinsights/${CLUSTER_NAME}/myservice
        log_stream_prefix   ${HOST_NAME}-
        auto_create_group   true
```

The next example removes an existing `conf` file entirely from `extraFiles`. This excludes the `application-log.conf` entirely by overriding it with an empty string. Simply omitting `application-log.conf` from `extraFiles` would instead imply to use the default, which is not what you are trying to achieve in this example. The same applies to removing any custom `conf` file that you might have previously added to `extraFiles`.

```
containerLogs:
  fluentBit:
config:
  extraFiles:
    application-log.conf: ""
```

Manage Kubernetes tolerations for the installed pod workloads

Starting with version 1.7.0 of the Amazon CloudWatch Observability EKS add-on, the add-on and the Helm chart by default set Kubernetes *tolerations* to tolerate all taints on the pod workloads that are installed by the add-on or the Helm chart. This ensures that daemonsets such as the CloudWatch agent and Fluent Bit can schedule pods on all nodes in your cluster by default. For more information about tolerations and taints, see Taints and Tolerations in the Kubernetes documentation.

The default tolerations set by the add-on or the Helm chart are as follows:

```
tolerations:
- operator: Exists
```

You can override the default tolerations by setting the `tolerations` field at the root level when using the add-on advanced config or when you install or upgrade the Helm chart with value overrides. An example would look like the following:

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

To omit tolerations completely, you can use a config that looks like the following:

```
tolerations: []
```

Any changes to tolerations apply to all pod workloads that are installed by the add-on or the Helm chart.

Opt out of accelerated compute metrics collection

By default, Container Insights with enhanced observability collects metrics for Accelerated Compute monitoring, including NVIDIA GPU metrics, AWS Neuron metrics for AWS Trainium and AWS Inferentia, and AWS Elastic Fabric Adapter (EFA) metrics.

NVIDIA GPU metrics from Amazon EKS workloads are collected by default beginning with version `v1.3.0-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300034.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see NVIDIA GPU metrics.

AWS Neuron metrics for AWS Trainium and AWS Inferentia accelerators are collected by default beginning with version `v1.5.0-eksbuild.1` of the EKS add-on or the Helm chart, and version `1.300036.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see AWS Neuron metrics for AWS Trainium and AWS Inferentia.

AWS Elastic Fabric Adapter (EFA) metrics from Linux nodes on Amazon EKS clusters are collected by default beginning with version `v1.5.2-eksbuild.1` of the EKS add-on or the Helm chart and version `1.300037.0` of the CloudWatch agent. For a list of metrics collected and prerequisites, see AWS Elastic Fabric Adapter (EFA) metrics.

You can opt out of collecting these metrics by setting the `accelerated_compute_metrics` field in the CloudWatch agent configuration file to `false`. This field is in the `kubernetes` section of the `metrics_collected` section in the CloudWatch configuration file. The following is an example of an opt-out configuration. For more information about how to use custom CloudWatch agent configurations, see the following section, Use a custom CloudWatch agent configuration.

```
{
  "logs": {
"metrics_collected": {
  "kubernetes": {
    "enhanced_container_insights": true,
    "accelerated_compute_metrics": false
  }
}
  }
}
```

Use a custom CloudWatch agent configuration

To collect other metrics, logs or traces using the CloudWatch agent, you can specify a custom configuration while also keeping Container Insights and CloudWatch Application Signals enabled. To do so, embed the CloudWatch agent configuration file within the config key under the agent key of the advanced configuration that you can use when creating or updating the EKS add-on or the Helm chart. The following represents the default agent configuration when you do not provide any additional configuration.

> **Important**
>
> Any custom configuration that you provide using additional configuration settings overrides the default configuration used by the agent. Be cautious not to unintentionally

disable functionality that is enabled by default, such as Container Insights with enhanced observability and CloudWatch Application Signals. In the scenario that you are required to provide a custom agent configuration, we recommend using the following default configuration as a baseline and then modifying it accordingly.

- For using the Amazon CloudWatch observability EKS add-on

```
--configuration-values '{
  "agent": {
"config": {
  "logs": {
    "metrics_collected": {
      "application_signals": {},
      "kubernetes": {
        "enhanced_container_insights": true
      }
    }
  },
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  }
}
  }
}'
```

- For using the Helm chart

```
--set agent.config='{
  "logs": {
"metrics_collected": {
  "application_signals": {},
  "kubernetes": {
    "enhanced_container_insights": true
  }
}
  },
  "traces": {
"traces_collected": {
  "application_signals": {}
```

```
}
  }
}'
```

The following example shows the default agent configuration for the CloudWatch agent on Windows. The CloudWatch agent on Windows does not support custom configuration.

```
{
  "logs": {
"metrics_collected": {
  "kubernetes": {
    "enhanced_container_insights": true
  },
}
  }
}
```

Manage admission webhook TLS certificates

The Amazon CloudWatch Observability EKS add-on and the Helm chart use Kubernetes admission webhooks to validate and mutate `AmazonCloudWatchAgent` and `Instrumentation` custom resource (CR) requests, and optionally Kubernetes pod requests on the cluster if CloudWatch Application Signals is enabled. In Kubernetes, webhooks require a TLS certificate that the API server is configured to trust to make sure secure communication.

By default, the Amazon CloudWatch Observability EKS add-on and the Helm chart auto-generate a self-signed CA and a TLS certificate signed by this CA for securing the communication between the API server and the webhook server. This auto-generated certificate has a default expiry of 10 years and is not auto-renewed upon expiry. In addition, the CA bundle and the certificate are re-generated every time the add-on or Helm chart is upgraded or re-installed, thus resetting the expiry. If you want to change the default expiry of the auto-generated certificate, you can use the following additional configurations when creating or updating the add-on. Replace *`expiry-in-days`* with your desired expiry duration in days.

- Use this for the Amazon CloudWatch Observability EKS add-on

```
--configuration-values '{ "admissionWebhooks": { "autoGenerateCert":
 { "expiryDays": expiry-in-days } } }'
```

- Use this for the Helm chart

```
--set admissionWebhooks.autoGenerateCert.expiryDays=expiry-in-days
```

For a more secure and feature-rich certificate authority solution, the add-on has opt-in support for cert-manager, a widely-adopted solution for TLS certificate management in Kubernetes that simplifies the process of obtaining, renewing, managing and using those certificates. It ensures that certificates are valid and up to date, and attempts to renew certificates at a configured time before expiry. cert-manager also simplifies issuing certificates from a variety of supported sources, including AWS Certificate Manager Private Certificate Authority.

We recommend that you review best practices for management of TLS certificates on your clusters and advise you to opt in to cert-manager for production environments. Note that if you opt-in to enabling cert-manager for managing the admission webhook TLS certificates, you are required to pre-install cert-manager on your Amazon EKS cluster before you install the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about available installation options, see cert-manager documentation. After you install it, you can opt in to using cert-manager for managing the admission webhook TLS certificates using the following additional configuration.

- If you are using the Amazon CloudWatch Observability EKS add-on

```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```

- If you are using the Helm chart

```
--set admissionWebhooks.certManager.enabled=true
```


```
--configuration-values '{ "admissionWebhooks": { "certManager": { "enabled":
 true } } }'
```


The advanced configuration discussed in this section will by default use a SelfSigned issuer.

Collect Amazon EBS volume IDs

If you want to collect Amazon EBS volume IDs in the performance logs, you must add another policy to the IAM role that is attached to the worker nodes or to the service account. Add the

following as an inline policy. For more information, see Adding and Removing IAM Identity Permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolumes"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Collect Java Management Extensions (JMX) metrics

The CloudWatch agent supports Java Management Extensions (JMX) metrics collection on Amazon EKS. This allows you to collect additional metrics from Java applications running on Amazon EKS clusters enabling insight into performance, memory usage, traffic, and other critical metrics. For more information, see Collect Java Management Extensions (JMX) metrics.

Enable Kueue metrics

Beginning with version `v2.4.0-eksbuild.1` of the the CloudWatch Observability EKS add-on, Container Insights for Amazon EKS supports collecting Kueue metrics from Amazon EKS clusters. For more information about these metrics, see Kueue metrics.

If you are using the Amazon SageMaker AI Hyperpod Task Governance EKS add-on, you can skip the steps in the **Prerequisites** section and just follow the steps in Enable the configuration flag.

Prerequisites

Before you install Kueue in your Amazon EKS cluster, make the following updates in the manifest file:

1. Enable the optional cluster queue resource metrics for Kueue. To do this, modify the in-line `controller_manager_config.yaml` in the `kueue-system` ConfigMap. In the `metrics` section, add or uncomment the line `enableClusterQueueResources: true`.

```
apiVersion: v1
data:
  controller_manager_config.yaml: |
apiVersion: config.kueue.x-k8s.io/v1beta1
kind: Configuration
health:
  healthProbeBindAddress: :8081
metrics:
  bindAddress: :8080
  enableClusterQueueResources: true  <-- ADD/UNCOMMENT THIS LINE
```

2. By default, all `k8s` services are available cluster-wide. Kueue creates a service `kueue-controller-manager-metrics-service` for exposing metrics. To prevent duplicate observations for metrics, modify this service to allow access only to the metrics service from the same node. To do this, add the line `internalTrafficPolicy: Local` to the `kueue-controller-manager-metrics-service` definition.

```
apiVersion: v1
kind: Service
metadata:
  labels:
...
  name: kueue-controller-manager-metrics-service
  namespace: kueue-system
spec:
  ports:
  - name: https
port: 8443
protocol: TCP
targetPort: https
  internalTrafficPolicy: Local   <-- ADD THIS LINE
  selector:
control-plane: controller-manager
```

3. Lastly, the `kueue-controller-manager` pod creates a `kube-rbac-proxy` container. This container currently has a high level of logging verbosity, which causes the cluster's bearer token to be logged by that container when the metrics scraper accesses the `kueue-controller-`

`manager-metrics-service`. We recommend that you decrease this logging verbosity. The default value in the manifest distributed by Kueue is 10, we recommend to change it to 0.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
...
  name: kueue-controller-manager
  namespace: kueue-system
spec:
  ...
  template:
...
spec:
  containers:
  ...
  - args:
    - --secure-listen-address=0.0.0.0:8443
    - --upstream=http://127.0.0.1:8080/
    - --logtostderr=true
    - --v=0  <-- CHANGE v=10 TO v=0
    image: gcr.io/kubebuilder/kube-rbac-proxy:v0.8.0
    name: kube-rbac-proxy
    ...
```

Enable the configuration flag

To enable the Kueue metrics, you must enable `kueue_container_insights` in the add-on additional configuration. You can do this either by using the AWS CLI to set up the EKS Observability add-on, or by using the Amazon EKS console.

After you have successfully installed the EKS Observability add-on with one of the following methods, you can view your Amazon EKS cluster metrics under the HyperPod console **Dashboard** tab.

AWS CLI

To enable Kueue metrics using the AWS CLI

- Enter the following AWS CLI command to install the add-on.

```
aws eks create-addon --cluster-name cluster-name --addon-name amazon-cloudwatch-
observability --configuration-values "configuration_json_file"
```

The following is an example of the JSON file with the configuration values.

```
{
"agent": {
    "config": {
        "logs": {
            "metrics_collected": {
                "kubernetes": {
                    "kueue_container_insights": true,
                    "enhanced_container_insights": true
                },
                "application_signals": { }
            }
        },
        "traces": {
            "traces_collected": {
                "application_signals": { }
            }
        }
    },
},
}
```

Amazon EKS console

To enable Kueue metrics using the Amazon EKS console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of your cluster.
3. Choose **Add-ons**.
4. Find the **Amazon CloudWatch Observability** add-on in the list, and install it. When you do so, choose **Optional configuration** and include the following JSON configuration values.

```
{
"agent": {
    "config": {
```

```
        "logs": {
            "metrics_collected": {
                "kubernetes": {
                    "kueue_container_insights": true,
                    "enhanced_container_insights": true
                },
                "application_signals": { }
            }
        },
        "traces": {
            "traces_collected": {
                "application_signals": { }
            }
        }
    },
},
}
```

Appending OpenTelemetry collector configuration files

The CloudWatch agent supports supplemental OpenTelemetry collector configuration files alongside its own configuration files. This feature allows you to use CloudWatch agent features such as CloudWatch Application Signals or Container Insights through the CloudWatch agent configuration and bring in your existing OpenTelemetry collector configuration with a single agent.

To prevent merge conflicts with pipelines automatically created by CloudWatch agent, add a custom suffix to each of the components and pipelines in your OpenTelemetry collector configuration. This will prevent clashing and merge conflicts.

- If you are using the Amazon CloudWatch Observability EKS add-on

```
--configuration-values file://values.yaml
```

or

```
--configuration-values '
  agent:
otelConfig:
  receivers:
    otlp/custom-suffix:
      protocols:
```

```
        http: {}
  exporters:
    awscloudwatchlogs/custom-suffix:
      log_group_name: "test-group"
      log_stream_name: "test-stream"
  service:
    pipelines:
      logs/custom-suffix:
        receivers: [otlp/custom-suffix]
        exporters: [awscloudwatchlogs/custom-suffix]
'
```

- If you are using the Helm chart

```
--set agent.otelConfig='
  receivers:
otlp/custom-suffix:
  protocols:
    http: {}
  exporters:
awscloudwatchlogs/custom-suffix:
  log_group_name: "test-group"
  log_stream_name: "test-stream"
  service:
pipelines:
  logs/custom-suffix:
    receivers: [otlp/custom-suffix]
    exporters: [awscloudwatchlogs/custom-suffix]
'
```

Enabling APM through Application Signals for your Amazon EKS cluster

By default, OpenTelemetry (OTEL) based Application Performance Monitoring (APM) is enabled through Application Signals when installing either the CloudWatch Observability EKS add-on (V5.0.0 or greater) or the Helm chart. You can further customize specific settings using the advanced configuration for the Amazon EKS add-on or by overriding values with the Helm chart.

> **Note**
>
> If you use any OpenTelemetry (OTEL) based APM solution, enabling Application Signals affects your existing observability setup. Review your current implementation before

proceeding. To maintain your existing APM setup after upgrading to V5.0.0 or later, see Opt out of Application Signals.

Application Signals Auto monitor

Version 5.0.0 of the CloudWatch Observability Amazon EKS add-on and Helm chart introduces new functionality. You can now automatically enable Application Signals for all or specific service workloads in your EKS cluster through the Auto monitor configuration. The following `autoMonitor` settings can be specified within the `applicationSignals` section under the `manager` section of the advanced configuration.

- *monitorAllServices* – A boolean flag to enable (true) or disable (false) monitoring of all service workloads by Auto monitor. Defaults to true. Enabling this flag will make sure that all Kubernetes workloads (Deployments, DaemonSets, and StatefulSets) in the cluster that are mapped to a Kubernetes Service will be in scope for automatic enablement of Application Signals when they are brought up for the first time (or when restarted for existing workloads). The system excludes workloads in the `kube-system` and `amazon-cloudwatch` namespaces by default.
- *languages* – A list of strings specifying the set of languages that Application Signals will try to automatically instrument your services with, when `monitorAllServices` is enabled. Defaults to all the supported languages.
- *restartPods* – A boolean flag controls whether workloads restart after configuration changes. Defaults to false. Enabling this flag to `true` controls whether Kubernetes workloads within Auto monitor scope restart automatically when saving configuration changes. Any settings on your Kubernetes workloads that influence the restart of the pods such as `updateStrategy` will be considered. Consider that restarting may cause service downtime.
- *customSelector* – Settings to select specific Kubernetes namespaces or workloads for Auto monitor.
  - *java* – Specify workloads to automatically instrument with Java
  - *python* – Specify workloads to automatically instrument with Python
  - *nodejs* – Specify workloads to automatically instrument with Node.js
  - *dotnet* – Specify workloads to automatically instrument with .NET

  For each of the languages above, the following fields can be configured.

- *namespaces* – A list of strings specifying the namespaces to be selected. Defaults to an empty list, that is []
- *deployments* – A list of strings specifying the deployments to be selected. Specify in `namespace/deployment`format. Defaults to an empty list, that is []
- *daemonsets* – A list of strings specifying the daemonsets to be selected. Specify in `namespace/daemonset` format. Defaults to an empty list, that is []
- *statefulsets* – A list of strings specifying the statefulsets to be selected. Specify in `namespace/statefulset` format. Defaults to an empty list, that is []

- *exclude* – Settings to exclude specific Kubernetes namespaces or workloads from Auto monitor. Excluding a workload takes precedence when the same workload is also within scope of `monitorAllServices` or `customSelector`.
  - *java* – Specify workloads to exclude from being automatically instrumented with Java
  - *python* – Specify workloads to exclude from being automatically instrumented with Python
  - *nodejs* – Specify workloads to exclude from being automatically instrumented with Node.js
  - *dotnet* – Specify workloads to exclude from being automatically instrumented with .NET

  For each of the languages above, the following fields can be configured.
  - *namespaces* – A list of strings specifying the namespaces to be excluded. Defaults to an empty list, that is []
  - *deployments* – A list of strings specifying the deployments to be excluded. Specify in `namespace/deployment`format. Defaults to an empty list, that is []
  - *daemonsets* – A list of strings specifying the daemonsets to be excluded. Specify in `namespace/daemonset` format. Defaults to an empty list, that is []
  - *statefulsets* – A list of strings specifying the statefulsets to be excluded. Specify in `namespace/statefulset` format. Defaults to an empty list, that is []

The following is an example configuration that automatically enables Application Signals for all existing and new service workloads on the cluster.

```
manager:
  applicationSignals:
autoMonitor:
  monitorAllServices: true
  restartPods: true
```

The following is an example configuration that automatically enables Application Signals for any new service workload that is brought up and for any existing service workload that is explicitly restarted on the cluster.

```
manager:
  applicationSignals:
autoMonitor:
  monitorAllServices: true
```

The following is an example configuration that automatically enables Application Signals with Java for all existing and new pods corresponding to a workload in the `pet-warehouse` namespace.

```
manager:
  applicationSignals:
autoMonitor:
  restartPods: true
  customSelector:
    java:
      namespaces: ["pet-warehouse"]
```

The following is an example configuration that automatically enables Application Signals with Python for all existing and new service workloads in the cluster excluding the `pet-clinic` deployment.

```
manager:
  applicationSignals:
autoMonitor:
  monitorAllServices: true
  languages: ["python"]
  restartPods: true
  exclude:
    python:
      deployments: ["pet-warehouse/pet-clinic"]
```

The following is an example configuration that automatically enables Application Signals with Java for all service workloads in the cluster except for the ones in the `python-apps` namespace and further enables Application Signals with Python specifically for the `sample-python-app` deployment in the `python-apps` namespace.

```
manager:
```

```
  applicationSignals:
autoMonitor:
  monitorAllServices: true
  languages: ["java"]
  restartPods: true
  customSelector:
    python:
      deployments: ["python-apps/sample-python-app"]
  exclude:
    java:
      namespaces: ["python-apps"]
```

Considerations for large Kubernetes clusters

If you run large Kubernetes clusters, you might need additional configuration to make sure the CloudWatch agent runs reliably. The following sections describe common issues and recommended configurations for large clusters.

Separate agent installations for cluster-level and node-level metrics

> **Note**
>
> This section applies only if you have Container Insights with enhanced observability enabled. If you use Container Insights with OpenTelemetry metrics exclusively, the default installation already deploys separate agent installations.

By default, the CloudWatch agent runs as a daemonset on each node in your cluster. One agent Pod is elected as a leader to collect cluster-level metrics. These metrics include control plane metrics and workload status metrics. On large clusters, this leader Pod may be terminated with an out-of-memory (OOM) error because it performs additional work while sharing the same resource limits as all other agent Pods.

To determine if you are experiencing this issue, check for agent Pods in a failed state with exit code `137` and reason `OOMKilled`. A common symptom is that one agent Pod (the leader) crashes, a new leader is elected, and that Pod also crashes, creating a recurring cycle. To confirm this behavior, inspect the leader election ConfigMap by running the following command.

```
kubectl describe configmap cwagent-clusterleader -n amazon-cloudwatch
```

If the ConfigMap shows frequent changes to the leader entry, this indicates that a new leader is being elected repeatedly, which confirms the issue.

To avoid this issue, you can separate the agent into two installations:

- A daemonset for node-level metrics and Application Signals
- A deployment for cluster-level metrics

This separation lets you manage and scale each installation independently.

To enable this configuration, use the advanced configuration to define two entries in the `agents` section:

1. Set the `CWAGENT_ROLE` environment variable to `NODE` on the daemonset agent.
2. Set the `CWAGENT_ROLE` environment variable to `LEADER` on the deployment agent.

> **Important**
>
> The deployment agent configuration must enable only Container Insights. Don't enable Application Signals on the deployment agent. Both installations run with `hostNetwork` by default, and enabling Application Signals on both causes port binding conflicts.

The following example shows an advanced configuration that separates the agent into two installations.

```
agents:
  - name: cloudwatch-agent
env:
  - name: CWAGENT_ROLE
    value: NODE
  - name: cloudwatch-agent-ci-leader
mode: deployment
config:
  {
    "logs": {
      "metrics_collected": {
        "kubernetes": {
          "enhanced_container_insights": true
```

```
        }
      }
    }
  }
env:
  - name: CWAGENT_ROLE
    value: LEADER
resources:
  limits:
    memory: 10Gi
    cpu: 2000m
```

In this example, besides setting the `CWAGENT_ROLE` environment variable on each agent installation, it overrides the default mode, configuration, and resource limits for the deployment agent only. The base `cloudwatch-agent` installation uses the defaults.

You can further customize each agent installation by adding `nodeSelector`, `tolerations`, or `nodeAffinity` fields to schedule the leader Pod on a specific node. The resource limits shown in the preceding example are illustrative. Adjust these values based on your cluster size and the number of metrics that the leader Pod collects.

Considerations for Amazon EKS Hybrid Nodes

Node-level metrics aren't available for hybrid nodes because Container Insights depends on the availability of the EC2 Instance Metadata Service (IMDS) for node-level metrics. Cluster, workload, Pod, and container-level metrics are available for hybrid nodes.

After you install the add-on, patch the `amazoncloudwatchagents` resource to add the `RUN_WITH_IRSA` environment variable so the agent runs successfully on hybrid nodes:

```
kubectl patch amazoncloudwatchagents cloudwatch-agent -n amazon-
cloudwatch --type=json -p '[{"op":"add","path":"/spec/env/-","value":
{"name":"RUN_WITH_IRSA","value":"True"}}]'
```

Troubleshooting the Amazon CloudWatch Observability EKS add-on or the Helm chart

Use the following information to help troubleshoot problems with the Amazon CloudWatch Observability EKS add-on or the Helm chart

Topics

- Updating and deleting the Amazon CloudWatch Observability EKS add-on or the Helm chart

- Verify the version of the CloudWatch agent used by the Amazon CloudWatch Observability EKS add-on or the Helm chart
- Handling a ConfigurationConflict when managing the add-on or the Helm chart

Updating and deleting the Amazon CloudWatch Observability EKS add-on or the Helm chart

For instructions about updating or deleting the Amazon CloudWatch Observability EKS add-on, see Managing Amazon EKS add-ons. Use `amazon-cloudwatch-observability` as the name of the add-on.

To delete the Helm chart in a cluster, enter the following command.

```
helm delete amazon-cloudwatch-observability -n amazon-cloudwatch --wait
```

Verify the version of the CloudWatch agent used by the Amazon CloudWatch Observability EKS add-on or the Helm chart

The Amazon CloudWatch Observability EKS add-on and the Helm chart installs a custom resource of kind `AmazonCloudWatchAgent` that controls the behavior of the CloudWatch agent daemonset on the cluster, including the version of the CloudWatch agent being used. You can get a list of all the `AmazonCloudWatchAgent` custom resources installed on your cluster u by entering the following command:

```
kubectl get amazoncloudwatchagent -A
```

In the output of this command, you should be able to check the version of the CloudWatch agent. Alternatively, you can also describe the `amazoncloudwatchagent` resource or one of the `cloudwatch-agent-*` pods running on your cluster to inspect the image being used.

Handling a ConfigurationConflict when managing the add-on or the Helm chart

When you install or update the Amazon CloudWatch Observability EKS add-on or the Helm chart, if you notice a failure caused by existing resources, it is likely because you already have the CloudWatch agent and its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster.

The error displayed by the add-on will include `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`,

The error displayed by the Helm chart will be similar to `Error: INSTALLATION FAILED: Unable to continue with install and invalid ownership metadata.`.

When the add-on or the Helm chart tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on or Helm chart. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the add-on or Helm chart when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see Setup guide (AWS CLI) for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the AWS CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

Opt out of Application Signals

Fine-tune your service monitoring preferences in the CloudWatch console or with the SDK.

To disable Application Signals auto-monitoring, follow the procedure below:

Using CLI or SDK

The following configuration can be applied either as an advanced configuration to the EKS add-on or as a values override when using the helm chart.

```
{
  "manager": {
"applicationSignals": {
  "autoMonitor": {
    "monitorAllServices": false
  }
}
  }
```

```
}
```

Restart your services for the changes to take effect.

Using the console

Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

1. In the navigation pane, under **Application Signals (APM)**, choose **Services**.
2. Choose **Enable Application Signals** to view the enablement page.
3. Clear the **Auto-Monitor** checkbox for each service you don't want to monitor.
4. Restart your services for the changes to take effect.

Advanced configuration for OTel Container Insights on Amazon EKS

This topic covers advanced configuration scenarios for OTel Container Insights on Amazon EKS. Use these configurations to customize metric collection, filter logs, collect telemetry across accounts, add custom dimensions, and tune resource allocation for large clusters.

Prerequisites

Before you configure advanced settings, verify that you meet the following requirements.

- OTel Container Insights installed and in `ACTIVE` status on your Amazon EKS cluster
- Amazon EKS cluster running Kubernetes version 1.28 or later
- AWS CLI version 2.15.0 or later
- `kubectl` configured to communicate with your target cluster
- IAM permissions: `eks:UpdateAddon`, `eks:DescribeAddon`, and `iam:AttachRolePolicy` (required for cross-account configuration)

General configuration pattern

All advanced configurations follow the same pattern. You pass a JSON configuration to the `amazon-cloudwatch-observability` add-on by using the `aws eks update-addon` command.

```
aws eks update-addon \
  --cluster-name cluster-name \
```

```
  --addon-name amazon-cloudwatch-observability \
  --configuration-values 'JSON-configuration' \
  --resolve-conflicts OVERWRITE
```

> **Important**
>
> The `--resolve-conflicts OVERWRITE` flag replaces any existing add-on configuration. To preserve existing settings, merge them with your new configuration before you run the command.

Log filtering

You can reduce CloudWatch Logs costs by excluding logs that match specific criteria. Use log filtering to drop debug-level or verbose logs before the agent sends them to CloudWatch.

To configure log filtering

1. Run the following command to update the add-on with a log filter configuration. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{
    "otelContainerInsights": {
      "enabled": true
    },
    "agent": {
      "config": {
        "logs": {
          "metrics_collected": {
            "kubernetes": {
              "enhanced_container_insights": true
            }
          },
          "exclude_filters": [
            {
              "type": "log_level_filter",
              "log_level": "DEBUG"
            }
          ]
```

```
        }
      }
    }
  }' \
  --resolve-conflicts OVERWRITE
```

2. Verify that the add-on status is ACTIVE after the update.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status" \
  --output text
```

The exclude_filters configuration removes log entries that match the specified log level before the agent sends them to CloudWatch Logs. This reduces your log ingestion volume and associated costs.

For additional cost control, set a log retention policy on your log groups to automatically delete older logs. For more information, see Set log retention. Also consider using the CloudWatch Logs Infrequent Access log class for log groups that you query infrequently. For more information, see CloudWatch Logs log classes.

Multi-account collection

You can send telemetry from workload accounts to a central monitoring account by configuring cross-account IAM role assumption. This approach gives you a single view of metrics and logs from multiple Amazon EKS clusters across different AWS accounts.

To set up multi-account collection

To create the cross-account role in the monitoring account

1. In the central monitoring account, create an IAM role with a trust policy that allows the workload account to assume it. Replace the example account ID (111122223333) with the AWS account ID of the workload account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Attach the `CloudWatchAgentServerPolicy` managed policy to the cross-account role.

To configure the add-on for cross-account delivery

1. In the workload account, update the add-on to assume the cross-account role. Replace *`cluster-name`* with the name of your Amazon EKS cluster and *`monitoring-account-id`* with the AWS account ID of the central monitoring account.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{
    "otelContainerInsights": {
      "enabled": true
    },
    "agent": {
      "config": {
        "credentials": {
          "role_arn": "arn:aws:iam::monitoring-account-id:role/
CrossAccountCWObservabilityRole"
        }
      }
    }
  }' \
  --resolve-conflicts OVERWRITE
```

2. Verify that the add-on status is `ACTIVE` after the update.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status" \
  --output text
```

Custom metric dimensions

You can add custom dimensions derived from Kubernetes labels to your Container Insights metrics. Custom dimensions allow you to filter and group metrics at a finer granularity, such as by team, environment, or application tier.

To add custom dimensions from Kubernetes labels

1. Run the following command to configure custom dimensions. Replace *cluster-name* with the name of your Amazon EKS cluster and *label-key* with the Kubernetes label to use as a dimension.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{
    "otelContainerInsights": {
      "enabled": true
    },
    "agent": {
      "config": {
        "logs": {
          "metrics_collected": {
            "kubernetes": {
              "enhanced_container_insights": true,
              "metric_dimensions": {
                "custom_dimensions": ["label-key"]
              }
            }
          }
        }
      }
    }
  }' \
  --resolve-conflicts OVERWRITE
```

2. Verify that the add-on status is `ACTIVE` after the update.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status" \
```

```
  --output text
```

After the configuration takes effect, the specified Kubernetes labels appear as dimensions on your Container Insights metrics in CloudWatch.

Resource tuning for large clusters

For large clusters, you might need to increase the CPU and memory limits for the CloudWatch agent DaemonSet. The default resource allocation works well for small clusters, but larger clusters generate more telemetry data and require additional agent resources.

The following table provides sizing guidelines based on cluster size.

Cluster size	CPU request	CPU limit	Memory request	Memory limit
Small (20 nodes or fewer)	100m	200m	128Mi	256Mi
Medium (21–100 nodes)	200m	400m	256Mi	512Mi
Large (100+ nodes)	300m	500m	384Mi	768Mi
Extra-large (500+ nodes)	500m	1000m	512Mi	1Gi

To configure resource limits for the agent DaemonSet

1. Run the following command to set resource requests and limits. Replace *cluster-name* with the name of your Amazon EKS cluster and the resource values with values from the preceding table.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{
    "otelContainerInsights": {
      "enabled": true
    },
```

```
    "agent": {
      "resources": {
        "requests": {
          "cpu": "cpu-request",
          "memory": "memory-request"
        },
        "limits": {
          "cpu": "cpu-limit",
          "memory": "memory-limit"
        }
      }
    }
  }' \
  --resolve-conflicts OVERWRITE
```

2. Verify that the add-on status is ACTIVE and that agent pods restart with the new resource allocation.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status" \
  --output text
```

3. Confirm that the agent pods are running with the new resource limits.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent -o
 jsonpath='{.items[0].spec.containers[0].resources}'
```

Verify configuration changes

After you apply any advanced configuration, verify that the add-on is healthy and that agent pods restarted successfully.

To verify configuration changes

1. Check that the add-on status is ACTIVE. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks describe-addon \
  --cluster-name cluster-name \
```

```
  --addon-name amazon-cloudwatch-observability \
  --query "addon.{Status:status,ConfigValues:configurationValues}" \
  --output table
```

2. Verify that agent pods restarted and are in Running status.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

All agent pods must show Running status with a recent restart time.

Troubleshooting

Use the following guidance to resolve common issues with advanced configurations.

Add-on update fails with ConfigurationConflict

Symptom: The `aws eks update-addon` command returns a `ConfigurationConflict` error.

Cause: The JSON configuration that you provided is malformed or contains invalid keys.

Solution: Complete the following steps to resolve this issue.

1. Validate your JSON configuration by using a JSON linter or by running the following command.

```
echo 'your-json-configuration' | python3 -m json.tool
```

2. Verify that all configuration keys are valid for the `amazon-cloudwatch-observability` add-on.
3. Retry the update with the corrected JSON.

Cross-account metrics not appearing in the monitoring account

Symptom: After you configure multi-account collection, metrics don't appear in the central monitoring account.

Cause: The cross-account IAM role trust policy or permissions are incorrect.

Solution: Complete the following steps to resolve this issue.

1. Verify that the trust policy in the monitoring account allows the workload account to assume the role.

2. Verify that the cross-account role has the `CloudWatchAgentServerPolicy` managed policy attached.
3. Check the agent logs for `AssumeRole` errors.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
tail=100 | grep -i "AssumeRole\|AccessDenied"
```

4. Verify that the agent IAM role in the workload account has `sts:AssumeRole` permission for the cross-account role ARN.

Solutions

With Solutions, you can get pre-built observability for third-party Kubernetes tools that run on your Amazon EKS clusters. Each solution collects metrics from a specific tool and surfaces them in a dedicated dashboard within Container Insights.

Topics

- Collecting Karpenter metrics with OTel Container Insights
- Collecting KEDA metrics with OTel Container Insights

Collecting Karpenter metrics with OTel Container Insights

Karpenter is the recommended node autoscaler for Amazon EKS. Container Insights collects Karpenter metrics through Prometheus scraping. With Container Insights, you gain built-in visibility into node provisioning, scaling speed, and resource use. You do not need to set up Prometheus or Grafana manually.

Karpenter metrics help you answer operational questions such as:

- Why is my pod pending?
- How fast is Karpenter provisioning nodes?
- Is consolidation working effectively?

Prerequisites

Karpenter metrics collection is enabled by default when the following requirements are met.

- Amazon EKS cluster running Kubernetes 1.28 or later

- Karpenter installed and running in the cluster (default namespace: `kube-system`)
- OTel Container Insights enabled on the cluster (see Quick start: OTel Container Insights on Amazon EKS)
- The `amazon-cloudwatch-observability` add-on in `ACTIVE` status

Configuring Karpenter metrics collection

Karpenter metrics collection is enabled by default. If you already have OTel Container Insights enabled (`otelContainerInsights.enabled: true`) and Karpenter is running in your cluster, metrics start flowing automatically with no additional configuration.

If the Karpenter scrape target is not present in the cluster, the pipeline finds no pods and does nothing.

To disable Karpenter metrics collection, set `otelContainerInsights.solutions.karpenter.enabled` to `false`. To disable all solutions at once, set `otelContainerInsights.solutions.enabled` to `false`. The global toggle overrides individual solution settings. If `solutions.enabled` is `false`, no solutions collect metrics, even if their individual `enabled` flag is `true`.

The following Helm values show the default configuration.

```
otelContainerInsights:
  solutions:
    enabled: true
    karpenter:
      enabled: true
```

To disable Karpenter metrics using the Amazon EKS add-on, run the following AWS CLI command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"otelContainerInsights":{"solutions":{"karpenter":
{"enabled":false}}}}' \
  --resolve-conflicts OVERWRITE
```

To disable Karpenter metrics using the Helm chart, run the following command.

```
helm upgrade amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  -n amazon-cloudwatch \
  --reuse-values \
  --set otelContainerInsights.solutions.karpenter.enabled=false
```

After you enable Karpenter metrics, the CloudWatch agent finds Karpenter pods by the label `app.kubernetes.io/name=karpenter`. It scrapes the `/metrics` endpoint on the port named `http-metrics` (default 8080) every 30 seconds.

Karpenter metrics pipeline architecture

The Karpenter metrics pipeline uses the following data flow:

1. The Prometheus receiver discovers Karpenter pods using Kubernetes service discovery.
2. The receiver scrapes the `/metrics` endpoint on each Karpenter pod.
3. The OpenTelemetry processor chain adds Kubernetes and cloud resource labels to each metric.
4. The exporter sends the enriched metrics to CloudWatch through OTLP.

The Karpenter pipeline runs independently from other metric pipelines (API server, Kube State Metrics, cAdvisor). This means a failure in one pipeline does not affect the others. If the Karpenter scrape target goes down, other pipelines keep working.

> **Note**
>
> Karpenter metrics are cluster-scoped.

Viewing the Karpenter dashboard

After you turn on Karpenter metrics, you can view the dashboard in the CloudWatch console.

To view the Karpenter dashboard

1. Open the CloudWatch console at CloudWatch console.
2. In the navigation pane, choose **Container Insights**, then choose **Performance monitoring**.
3. For **Service**, choose **EKS OTEL**.
4. In the navigation pane, expand **Solutions** and choose **Karpenter**.

The dashboard displays 30 metrics organized into 8 categories: Pod Scheduling, Node Resources, Cluster State, NodePool, Cloud Provider, Disruption, Interruption, and Controller Health.

Available metrics

The following table lists the 30 Karpenter metrics that Container Insights collects.

Metric name	Category	Description
karpenter_pods_state	Pod Scheduling	Number of pods by state (phase label)
karpenter_pods_bound_duration_seconds	Pod Scheduling	Time for pods to reach bound state
karpenter_nodes_allocatable	Node Resources	Allocatable resources per node by resource type
karpenter_nodes_total_daemon_requests	Node Resources	Total resource requests from DaemonSet pods
karpenter_nodes_total_daemon_limits	Node Resources	Total resource limits from DaemonSet pods
karpenter_nodes_total_pod_requests	Node Resources	Total resource requests from all pods
karpenter_nodes_total_pod_limits	Node Resources	Total resource limits from all pods
karpenter_nodes_system_overhead	Node Resources	Resources reserved for system daemons
karpenter_nodes_current_lifetime_seconds	Node Resources	Current age of each node in seconds
karpenter_cluster_state_node_count	Cluster State	Number of nodes tracked in Karpenter cluster state

Metric name	Category	Description
karpenter_cluster_state_synced	Cluster State	Whether cluster state is synced (1=synced, 0=unsynced)
karpenter_cluster_state_unsynced_time_seconds	Cluster State	Duration cluster state has been unsynced
karpenter_cluster_utilization_percent	Cluster State	Percentage of allocatable resources used by pod requests
karpenter_nodepools_limit	NodePool	Configured resource limits per NodePool
karpenter_nodepools_usage	NodePool	Current resource usage per NodePool
karpenter_cloudprovider_duration_seconds	Cloud Provider	Cloud provider API call latency by method
karpenter_cloudprovider_instance_type_cpu_cores	Cloud Provider	Available vCPU cores per instance type
karpenter_cloudprovider_instance_type_memory_bytes	Cloud Provider	Available memory per instance type
karpenter_cloudprovider_instance_type_offering_available	Cloud Provider	Whether an instance type offering is available
karpenter_cloudprovider_instance_type_offering_price_estimate	Cloud Provider	Estimated hourly price per instance type offering

Metric name	Category	Description
karpenter_voluntary_disruption_eligible_nodes	Disruption	Nodes eligible for voluntary disruption by reason
karpenter_voluntary_disruption_consolidation_timeouts_total	Disruption	Number of consolidation evaluation timeouts
karpenter_voluntary_disruption_decision_evaluation_duration_seconds	Disruption	Time to evaluate disruption decisions
karpenter_nodes_created_total	Node Resources	Total nodes created by Karpenter
karpenter_nodes_terminated_total	Node Resources	Total nodes terminated by Karpenter (by reason)
karpenter_interruption_received_messages_total	Interruption	SQS interruption messages received (spot, rebalance, etc.)
karpenter_interruption_deleted_messages_total	Interruption	SQS interruption messages deleted/processed
controller_runtime_reconcile_errors_total	Controller Health	Total reconciliation errors across controllers
workqueue_depth	Controller Health	Current depth of the controller work queue
leader_election_master_status	Controller Health	Whether this instance is the elected leader (1=leader, 0=standby)

Collecting KEDA metrics with OTel Container Insights

KEDA (Kubernetes Event-Driven Autoscaling) is an open-source project that allows you to scale Kubernetes workloads based on external metrics from sources like Amazon SQS, Amazon CloudWatch, or custom Prometheus queries. Container Insights collects KEDA metrics through Prometheus scraping. With Container Insights, you gain built-in visibility into ScaledObject status, scaler health, and scale loop performance. You do not need to set up Prometheus or Grafana manually.

KEDA metrics help you answer operational questions such as:

- Are my ScaledObjects scaling correctly?
- Which scalers are experiencing errors?
- What is the current scale loop latency?

Prerequisites

KEDA metrics collection is enabled by default when the following requirements are met.

- Amazon EKS cluster running Kubernetes 1.28 or later
- KEDA installed and running in the cluster (default namespace: `keda`)
- OTel Container Insights enabled on the cluster (see Quick start: OTel Container Insights on Amazon EKS)
- The `amazon-cloudwatch-observability` add-on in `ACTIVE` status

Configuring KEDA metrics collection

KEDA metrics collection is enabled by default. If you already have OTel Container Insights enabled (`otelContainerInsights.enabled: true`) and KEDA is running in your cluster, metrics start flowing automatically with no additional configuration.

If the KEDA scrape target is not present in the cluster, the pipeline finds no pods and does nothing.

To disable KEDA metrics collection, set `otelContainerInsights.solutions.keda.enabled` to `false`. To disable all solutions at once, set `otelContainerInsights.solutions.enabled` to `false`. The global toggle overrides individual solution settings. If `solutions.enabled` is `false`, no solutions collect metrics, even if their individual `enabled` flag is `true`.

The following Helm values show the default configuration.

```
otelContainerInsights:
  solutions:
    enabled: true
    keda:
      enabled: true
```

To disable KEDA metrics using the Amazon EKS add-on, run the following AWS CLI command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"otelContainerInsights":{"solutions":{"keda":
{"enabled":false}}}}' \
  --resolve-conflicts OVERWRITE
```

To disable KEDA metrics using the Helm chart, run the following command.

```
helm upgrade amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  -n amazon-cloudwatch \
  --reuse-values \
  --set otelContainerInsights.solutions.keda.enabled=false
```

After you enable KEDA metrics, the CloudWatch agent finds KEDA operator pods by the label `app.kubernetes.io/name=keda-operator`. It scrapes the `/metrics` endpoint on the port named `metrics` every 30 seconds.

KEDA metrics pipeline architecture

The KEDA metrics pipeline uses the following data flow:

1. The Prometheus receiver discovers KEDA operator pods using Kubernetes service discovery.
2. The receiver scrapes the `/metrics` endpoint on each KEDA operator pod.
3. The OpenTelemetry processor chain adds Kubernetes and cloud resource labels to each metric.
4. The exporter sends the enriched metrics to CloudWatch through OTLP.

The KEDA pipeline runs independently from other metric pipelines (API server, Kube State Metrics, cAdvisor). This means a failure in one pipeline does not affect the others. If the KEDA scrape target goes down, other pipelines keep working.

> **Note**
>
> KEDA metrics are cluster-scoped.

Viewing the KEDA dashboard

After you turn on KEDA metrics, you can view the dashboard in the CloudWatch console.

To view the KEDA dashboard

1. Open the CloudWatch console at CloudWatch console.
2. In the navigation pane, choose **Container Insights**, then choose **Performance monitoring**.
3. For **Service**, choose **EKS OTEL**.
4. In the navigation pane, expand **Solutions** and choose **KEDA**.

The dashboard displays 10 metrics organized into 5 categories: Scaled Objects, Scalers, Triggers, Scaling, and Controller Health.

Available metrics

The following table lists the 10 KEDA metrics that Container Insights collects.

Metric name	Category	Description
`keda_scaled_object_paused`	Scaled Objects	1 if ScaledObject is paused, 0 otherwise
`keda_scaled_object_errors_total`	Scaled Objects	Total errors per ScaledObject
`keda_scaler_active`	Scalers	1 if scaler is active, 0 otherwise
`keda_scaler_metrics_value`	Scalers	Current value returned by the scaler

Metric name	Category	Description
keda_scaler_metrics_latency_seconds	Scalers	Latency of scaler metric retrieval
keda_scaler_detail_errors_total	Scalers	Total errors per scaler
keda_trigger_registered_total	Triggers	Total registered triggers by type
keda_internal_scale_loop_latency_seconds	Scaling	Scale loop latency per ScaledObject
keda_resource_registered_total	Scaling	Total KEDA custom resources registered by type
keda_build_info	Controller Health	KEDA build information (version, git commit)

Enhanced Container Insights (Classic)

Enhanced Container Insights (Classic) is a managed Amazon EKS add-on configuration that deploys the CloudWatch agent and Fluent Bit as a DaemonSet on your cluster. It provides Container Insights with Enhanced Observability, collecting infrastructure metrics, application logs, and performance data from your Amazon EKS workloads.

Both Enhanced Container Insights (Classic) and OTel Container Insights use the same `amazon-cloudwatch-observability` Amazon EKS add-on. The difference is the add-on version and configuration. Enhanced Container Insights (Classic) refers to the legacy configuration of that add-on.

Maintenance mode

Enhanced Container Insights (Classic) is in maintenance mode. It continues to receive security patches and critical bug fixes, but no new features are being developed for this approach.

For new deployments, we recommend OTel Container Insights (Recommended), which provides broader signal coverage, simpler deployment, and active ongoing development.

This approach is designed for the following use cases:

- Existing customers who already have the add-on deployed and operational
- Teams that require a managed add-on model with Amazon EKS add-on lifecycle management

What Enhanced Container Insights (Classic) provides

Enhanced Container Insights (Classic) includes the following capabilities:

- **Enhanced Observability** — Detailed Kubernetes metrics at the pod, node, and cluster level
- **Metrics and logs** — Collects infrastructure metrics and container logs through the CloudWatch agent and Fluent Bit
- **Managed lifecycle** — Install, update, and remove through the Amazon EKS add-on API or console
- **Automatic updates** — Security patches delivered through the Amazon EKS add-on update mechanism

Comparison with OTel Container Insights

The following table summarizes the differences between Enhanced Container Insights (Classic) and OTel Container Insights.

Consideration	Enhanced Container Insights (Classic)	OTel Container Insights (Recommended)
Signals	Metrics, Logs	Metrics, Logs
Enhanced Observability	Yes	Yes
Maintenance status	Maintenance mode	Active development
Deployment model	Amazon EKS managed add-on	Amazon EKS add-on (OTel-based)

Installation methods

You can install Enhanced Container Insights (Classic) using any of the following methods:

- **Amazon EKS console** — Enable from the **Observability** tab on your cluster. For more information, see Enable Enhanced Container Insights (Classic) from the console.
- **AWS CLI** — Use the `aws eks create-addon` command with the `amazon-cloudwatch-observability` add-on name and a v5.x version. For more information, see Setup guide (AWS CLI).
- **Helm chart** — Use `helm install` with a Classic-compatible version of the chart.
- **CloudFormation or AWS CDK** — Use `AWS::EKS::Addon` with the version pinned to a v5.x release. For more information, see Deploy Enhanced Container Insights (Classic) with CloudFormation or Helm.

Metrics reference

For the complete list of Enhanced Container Insights (Classic) metric names and dimensions, see Enhanced Container Insights (Classic) metric reference.

Upgrade to OTel Container Insights

For step-by-step instructions to move from Enhanced Container Insights (Classic) to OTel Container Insights with zero monitoring gaps, see Migrate from Enhanced Container Insights (Classic) to OTel Container Insights.

Topics

- Setup guide (AWS CLI)
- Enable Enhanced Container Insights (Classic) from the console
- Deploy Enhanced Container Insights (Classic) with CloudFormation or Helm
- Enhanced Container Insights (Classic) metric reference
- Send logs to CloudWatch Logs

Setup guide (AWS CLI)

This guide walks you through installing Enhanced Container Insights (Classic) on an existing Amazon EKS cluster by using the AWS CLI. The add-on deploys a CloudWatch agent as a DaemonSet that collects infrastructure metrics, container logs, and performance data.

Maintenance mode

Enhanced Container Insights (Classic) is in maintenance mode. For new deployments, you should recommend OTel Container Insights (Recommended).

Prerequisites

Before you begin, verify that you meet the following requirements.

- An existing Amazon EKS cluster running Kubernetes version 1.25 or later
- Platform version `eks.1` or later
- AWS CLI version 2.12.0 or later
- `kubectl` configured to communicate with your target cluster
- IAM permissions: `eks:CreateAddon`, `eks:DescribeAddon`, `iam:CreateServiceLinkedRole`, `iam:CreateRole`, and `iam:AttachRolePolicy`
- The EKS Pod Identity Agent add-on installed on your cluster, or IAM Roles for Service Accounts (IRSA) configured
- Outbound internet access from the cluster to CloudWatch endpoints

Step 1: Create the IAM role

Create an IAM role that allows the CloudWatch agent to send data to CloudWatch.

To create the CloudWatch agent IAM role

1. Run the following command to create the role with a trust policy for EKS Pod Identity.

```
aws iam create-role \
  --role-name EKS-CloudWatch-Agent-Role \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }]
  }'
```

2. Attach the `CloudWatchAgentServerPolicy` managed policy to the role.

```
aws iam attach-role-policy \
  --role-name EKS-CloudWatch-Agent-Role \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

Step 2: Create the Pod Identity association

Associate the IAM role with the CloudWatch agent service account in your cluster.

To create the Pod Identity association

- Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster and *`account-id`* with your AWS account ID.

```
aws eks create-pod-identity-association \
  --cluster-name cluster-name \
  --namespace amazon-cloudwatch \
  --service-account cloudwatch-agent \
  --role-arn arn:aws:iam::account-id:role/EKS-CloudWatch-Agent-Role
```

Step 3: Install the Amazon CloudWatch Observability add-on

Install the `amazon-cloudwatch-observability` add-on on your cluster.

To install the add-on

- Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

```
aws eks create-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --addon-version v5.x.x-eksbuild.x
```

  Replace *`v5.x.x-eksbuild.x`* with the latest available v5.x version. Installing without `--addon-version` installs the latest version (v6.2.0+), which activates the OTel pipeline instead of Classic.

Step 4: Verify the add-on status

Confirm that the add-on installed successfully.

To verify the add-on status

- Run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status" \
  --output text
```

  The expected output is `ACTIVE`.

Step 5: Confirm agent pods are running

Verify that the CloudWatch agent pods are running in the `amazon-cloudwatch` namespace.

To confirm agent pods are running

- Run the following command.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

  All agent pods must show `Running` status.

Verify data in CloudWatch

After you complete the setup, Container Insights data appears in CloudWatch within 3 to 5 minutes.

To verify Container Insights data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Container Insights**.

3. Verify that metrics such as `node_cpu_utilization` and `pod_memory_utilization` appear for your cluster.

Troubleshooting

Use the following guidance to resolve common issues when you install Enhanced Container Insights (Classic) on Amazon EKS.

Add-on status shows CREATE_FAILED or DEGRADED

Symptom: When you run `aws eks describe-addon`, the status shows `CREATE_FAILED` or `DEGRADED`.

Cause: The add-on installation failed because of insufficient IAM permissions or a missing Pod Identity association.

Solution: Complete the following steps to resolve this issue.

1. Run the following command to check for detailed error information. Replace *`cluster-name`* with the name of your cluster.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.health"
```

2. Verify that the IAM role exists and has the `CloudWatchAgentServerPolicy` attached.
3. Verify that the Pod Identity association targets the correct namespace (`amazon-cloudwatch`) and service account (`cloudwatch-agent`).
4. Delete the failed add-on and reinstall it after you resolve the issue.

```
aws eks delete-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability
```

Metrics not appearing in CloudWatch after 5 minutes

Symptom: The agent pods show `Running` status, but no metrics appear in CloudWatch after 5 minutes.

Cause: The agent can't send data to CloudWatch because of network restrictions or incorrect IAM permissions.

Solution: Complete the following steps to resolve this issue.

1. Verify that your VPC security groups and network ACLs allow outbound HTTPS traffic (port 443) to CloudWatch endpoints.
2. Check the agent logs for permission errors or connection timeouts.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --
tail=100 | grep -i "error\|timeout\|denied"
```

3. Verify that the IAM role has the `CloudWatchAgentServerPolicy` policy attached and that the trust policy allows `pods.eks.amazonaws.com`.

Agent pods stuck in Pending or CrashLoopBackOff

Symptom: When you run `kubectl get pods -n amazon-cloudwatch`, one or more pods show `Pending` or `CrashLoopBackOff` status.

Cause: The agent pods can't start because of insufficient node resources, image pull failures, or missing permissions.

Solution: Complete the following steps to resolve this issue.

1. Check the pod events for detailed error messages.

```
kubectl describe pod -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

2. Verify that your nodes have sufficient CPU and memory available for the agent pods.
3. Verify that the EKS Pod Identity Agent add-on is installed and running.

```
kubectl get pods -n kube-system -l app.kubernetes.io/name=eks-pod-identity-agent
```

Next steps

Ready to upgrade? For step-by-step instructions to move from Enhanced Container Insights (Classic) to OTel Container Insights, see Migrate from Enhanced Container Insights (Classic) to OTel Container Insights.

Enable Enhanced Container Insights (Classic) from the console

You can enable Enhanced Container Insights (Classic) directly from the Amazon EKS console. This approach installs the Amazon CloudWatch Observability add-on (version prior to v6.2.0) with the Classic metric pipeline. This is ideal if you prefer a UI-driven experience and want to enable observability without writing infrastructure code or running CLI commands.

> **Recommendation**
>
> For new deployments, we recommend OTel Container Insights (Recommended), which uses add-on version 6.2.0 or later with the OTel-based pipeline.

Prerequisites

Before you enable Enhanced Container Insights (Classic) from the console, verify that you meet the following requirements.

- You are signed in to the Amazon EKS console.
- An existing Amazon EKS cluster running Kubernetes version 1.25 or later.
- The `amazon-cloudwatch-observability` add-on version must be prior to v6.2.0 (for example, v5.4.0).
- IAM permissions: `eks:DescribeCluster`, `eks:UpdateClusterConfig`, `eks:ListAddons`, `eks:CreateAddon`, `eks:DescribeAddon`, `iam:AttachRolePolicy`, and `iam:CreateServiceLinkedRole`.
- The node IAM role must have the `CloudWatchAgentServerPolicy` managed policy attached.
- The cluster is in `ACTIVE` state.

Enable Enhanced Container Insights (Classic)

Use the following procedure to enable Enhanced Container Insights (Classic) from the Amazon EKS console.

To enable Enhanced Container Insights (Classic) by using the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/.
2. Choose **Clusters**, and then choose your cluster name.

3. Choose the **Observability** tab.
4. Next to **Container Insights** status, choose **Manage**.
5. Select **Enhanced Container Insights**, and then choose **Enable**.

The console installs the add-on with the Classic configuration. Metrics appear in the `ContainerInsights` namespace within 3 to 5 minutes.

Verify that data appears in CloudWatch

After you enable Enhanced Container Insights (Classic), data appears in CloudWatch within 3 to 5 minutes.

To verify Container Insights data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**.
3. Choose the **ContainerInsights** namespace.
4. Verify that metrics such as `pod_cpu_utilization` and `node_memory_utilization` appear for your cluster.

Next steps

After you enable Enhanced Container Insights (Classic), you can explore the following topics.

- For AWS CLI and advanced setup options, see Setup guide (AWS CLI).
- For information about the metrics that Enhanced Container Insights (Classic) collects, see Enhanced Container Insights (Classic) metric reference.
- To migrate to OTel Container Insights, see Migrate from Enhanced Container Insights (Classic) to OTel Container Insights.
- For more information about OTel Container Insights, see OTel Container Insights (Recommended).

Deploy Enhanced Container Insights (Classic) with CloudFormation or Helm

You can deploy Enhanced Container Insights (Classic) as infrastructure as code by using CloudFormation templates or Helm charts. This approach installs the `amazon-cloudwatch-`

`observability` add-on with a version prior to v6.2.0 to enable the Classic metric pipeline. Choose this approach if you manage clusters through IaC pipelines and want repeatable, version-controlled deployments.

> **Recommendation**
>
> For new deployments, we recommend OTel Container Insights (Recommended), which uses add-on version 6.2.0 or later with the OTel-based pipeline.

Prerequisites

Before you deploy Enhanced Container Insights (Classic) with CloudFormation or Helm, verify that you meet the following requirements.

- An existing Amazon EKS cluster running Kubernetes version 1.25 or later
- AWS CLI version 2.12.0 or later
- `kubectl` configured to communicate with your target cluster
- IAM permissions for CloudFormation stack creation (for the CloudFormation method)
- Helm v3.8 or later (for the Helm method)
- Outbound internet access from the cluster to CloudWatch endpoints

Deploy with CloudFormation

Use a CloudFormation template to deploy the CloudWatch Observability add-on with the Classic configuration. This template uses the `AWS::EKS::Addon` resource to install the add-on with a version prior to v6.2.0.

CloudFormation template

The following template creates the resources that you need to enable Enhanced Container Insights (Classic) on your Amazon EKS cluster.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy Enhanced Container Insights (Classic) on an EKS cluster

Parameters:
  ClusterName:
```

```
    Type: String
    Description: The name of your EKS cluster

Resources:
  CloudWatchAgentRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: EKS-CloudWatch-Agent-Role
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: pods.eks.amazonaws.com
            Action:
              - sts:AssumeRole
              - sts:TagSession
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

  PodIdentityAssociation:
    Type: AWS::EKS::PodIdentityAssociation
    Properties:
      ClusterName: !Ref ClusterName
      Namespace: amazon-cloudwatch
      ServiceAccount: cloudwatch-agent
      RoleArn: !GetAtt CloudWatchAgentRole.Arn

  CloudWatchObservabilityAddon:
    Type: AWS::EKS::Addon
    DependsOn: PodIdentityAssociation
    Properties:
      ClusterName: !Ref ClusterName
      AddonName: amazon-cloudwatch-observability
      AddonVersion: v5.4.0-eksbuild.1
      ServiceAccountRoleArn: !GetAtt CloudWatchAgentRole.Arn
      ResolveConflicts: OVERWRITE
```

This template creates the following resources:

- An IAM role with the `CloudWatchAgentServerPolicy` managed policy attached
- An EKS Pod Identity association that maps the role to the CloudWatch agent service account
- The `amazon-cloudwatch-observability` EKS add-on with a Classic version (prior to v6.2.0)

To deploy the CloudFormation stack

Use the AWS CLI to create the CloudFormation stack from the template.

To deploy the stack

1. Save the preceding template to a file named `classic-container-insights.yaml`.
2. Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster.

```
aws cloudformation create-stack \
  --stack-name classic-container-insights \
  --template-body file://classic-container-insights.yaml \
  --parameters ParameterKey=ClusterName,ParameterValue=cluster-name \
  --capabilities CAPABILITY_NAMED_IAM
```

3. Wait for the stack creation to complete.

```
aws cloudformation wait stack-create-complete \
  --stack-name classic-container-insights
```

4. Verify that the stack status is `CREATE_COMPLETE`.

```
aws cloudformation describe-stacks \
  --stack-name classic-container-insights \
  --query "Stacks[0].StackStatus" \
  --output text
```

Deploy with Helm

Use the Helm chart to deploy Enhanced Container Insights (Classic) if you manage Kubernetes workloads with Helm. The chart installs the CloudWatch Observability operator with the Classic metric pipeline.

> **Note**
>
> When you use Helm, you must create the IAM role and Pod Identity association separately. The Helm chart installs only the Kubernetes resources.

Step 1: Create the IAM role

Create an IAM role for the CloudWatch agent before you install the Helm chart.

To create the IAM role and Pod Identity association

1. Run the following command to create the role with a trust policy for EKS Pod Identity.

```
aws iam create-role \
  --role-name EKS-CloudWatch-Agent-Role \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }]
  }'
```

2. Attach the `CloudWatchAgentServerPolicy` managed policy to the role.

```
aws iam attach-role-policy \
  --role-name EKS-CloudWatch-Agent-Role \
  --policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

3. Create the Pod Identity association. Replace *`cluster-name`* with the name of your Amazon EKS cluster and *`account-id`* with your AWS account ID.

```
aws eks create-pod-identity-association \
  --cluster-name cluster-name \
  --namespace amazon-cloudwatch \
  --service-account cloudwatch-agent \
  --role-arn arn:aws:iam::account-id:role/EKS-CloudWatch-Agent-Role
```

Step 2: Install the Helm chart

Install the CloudWatch Observability Helm chart with a Classic version (prior to v6.2.0).

To install the Helm chart

1. Run the following command. Replace *`cluster-name`* with the name of your Amazon EKS cluster and *`region`* with your AWS Region.

```
helm install amazon-cloudwatch-observability \
  oci://public.ecr.aws/cloudwatch-agent/amazon-cloudwatch-observability-helm-chart
 \
  --namespace amazon-cloudwatch \
  --create-namespace \
  --set clusterName=cluster-name \
  --set region=region \
  --version 5.4.0
```

2. Verify that the Helm release deployed successfully.

```
helm list -n amazon-cloudwatch
```

The status must show `deployed`.

Verify the deployment

After the CloudFormation stack creation or Helm installation completes, verify that the add-on is running and sending data to CloudWatch.

To verify the deployment

1. Confirm that the CloudWatch agent pods are running.

```
kubectl get pods -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent
```

All pods must show `Running` status.

2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the navigation pane, choose **Metrics**, and then choose **Classic metrics**.
4. Verify that metrics appear in the `ContainerInsights` namespace for your cluster.

Metrics typically appear in CloudWatch within 3 to 5 minutes after the deployment completes.

Next steps

After you deploy Enhanced Container Insights (Classic), you can explore the following topics.

- For console-based setup, see Enable Enhanced Container Insights (Classic) from the console.

- For AWS CLI-based setup, see Setup guide (AWS CLI).
- For information about the metrics that Enhanced Container Insights (Classic) collects, see Enhanced Container Insights (Classic) metric reference.
- To migrate to OTel Container Insights, see Migrate from Enhanced Container Insights (Classic) to OTel Container Insights.
- For more information about OTel Container Insights, see OTel Container Insights (Recommended).

Enhanced Container Insights (Classic) metric reference

Enhanced Container Insights (Classic) publishes metrics to the `ContainerInsights` namespace in CloudWatch. The following tables list all metrics by category, along with their dimensions and descriptions.

> **Note**
>
> For OTel Container Insights metrics, see OTel Container Insights (Recommended), which lists open-source metric names.

Cluster metrics

The following metrics measure the overall health and capacity of your Amazon EKS cluster.

Metric Name	Dimensions	Description
`cluster_failed_node_count`	`ClusterName`	The number of failed worker nodes in the cluster.
`cluster_node_count`	`ClusterName`	The total number of worker nodes in the cluster.

Node metrics

The following metrics measure resource usage and capacity at the node level.

Metric Name	Dimensions	Description
node_cpu_limit	ClusterName , InstanceId , NodeName	The maximum number of CPU units that can be assigned to the node.
node_cpu_reserved_capacity	ClusterName , InstanceId , NodeName	The percentage of CPU units reserved on the node.
node_cpu_usage_total	ClusterName , InstanceId , NodeName	The number of CPU units used on the node.
node_cpu_utilization	ClusterName , InstanceId , NodeName	The percentage of CPU units used on the node.
node_filesystem_utilization	ClusterName , InstanceId , NodeName	The percentage of filesystem capacity used on the node.
node_memory_limit	ClusterName , InstanceId , NodeName	The maximum amount of memory, in bytes, that can be assigned to the node.
node_memory_reserved_capacity	ClusterName , InstanceId , NodeName	The percentage of memory reserved on the node.
node_memory_utilization	ClusterName , InstanceId , NodeName	The percentage of memory used on the node.
node_memory_working_set	ClusterName , InstanceId , NodeName	The amount of memory, in bytes, in the working set of the node.
node_network_total_bytes	ClusterName , InstanceId , NodeName	The total number of bytes per second transmitted and received over the network on the node.
node_number_of_running_containers	ClusterName , InstanceId , NodeName	The number of running containers on the node.

Metric Name	Dimensions	Description
node_number_of_running_pods	ClusterName , InstanceId , NodeName	The number of running pods on the node.

Pod metrics

The following metrics measure resource usage at the pod level.

Metric Name	Dimensions	Description
pod_cpu_utilization	ClusterName , Namespace , PodName	The percentage of CPU units used by the pod.
pod_cpu_utilization_over_pod_limit	ClusterName , Namespace , PodName	The percentage of CPU units used relative to the pod limit.
pod_cpu_usage_total	ClusterName , Namespace , PodName	The number of CPU units used by the pod.
pod_memory_utilization	ClusterName , Namespace , PodName	The percentage of memory used by the pod.
pod_memory_utilization_over_pod_limit	ClusterName , Namespace , PodName	The percentage of memory used relative to the pod limit.
pod_memory_working_set	ClusterName , Namespace , PodName	The amount of memory, in bytes, in the working set of the pod.
pod_network_rx_bytes	ClusterName , Namespace , PodName	The number of bytes received per second over the network by the pod.
pod_network_tx_bytes	ClusterName , Namespace , PodName	The number of bytes transmitted per second over the network by the pod.

Metric Name	Dimensions	Description
pod_number_of_container_restarts	ClusterName , Namespace , PodName	The total number of container restarts in the pod.

Container metrics

The following metrics measure resource usage at the container level.

Metric Name	Dimensions	Description
container_cpu_utilization	ClusterName , Namespace , PodName, ContainerName	The percentage of CPU units used by the container.
container_memory_utilization	ClusterName , Namespace , PodName, ContainerName	The percentage of memory used by the container.

Service and namespace metrics

The following metrics measure pod counts at the service and namespace level.

Metric Name	Dimensions	Description
service_number_of_running_pods	ClusterName , Namespace , Service	The number of running pods for the service.
namespace_number_of_running_pods	ClusterName , Namespace	The number of running pods in the namespace.

NVIDIA GPU metrics

The following metrics are collected when your cluster has NVIDIA GPU-equipped nodes. These metrics help you monitor GPU compute and memory utilization across nodes, pods, and containers.

Metric Name	Dimensions	Description
node_gpu_utilization	ClusterName , InstanceId , NodeName	The percentage of GPU compute capacity used on the node.
node_gpu_memory_utilization	ClusterName , InstanceId , NodeName	The percentage of GPU memory used on the node.
node_gpu_temperature	ClusterName , InstanceId , NodeName	The GPU temperature in degrees Celsius on the node.
pod_gpu_utilization	ClusterName , Namespace , PodName	The percentage of GPU compute capacity used by the pod.
container_gpu_utilization	ClusterName , Namespace , PodName, ContainerName	The percentage of GPU compute capacity used by the container.

AWS Neuron metrics

The following metrics are collected when your cluster has AWS Neuron-equipped instances, such as Inf1 or Inf2. These metrics help you monitor NeuronCore utilization at the node and pod level.

Metric Name	Dimensions	Description
node_neuroncore_utilization	ClusterName , InstanceId , NodeName	The percentage of NeuronCore compute capacity used on the node.
pod_neuroncore_utilization	ClusterName , Namespace , PodName	The percentage of NeuronCore compute capacity used by the pod.

Kubernetes API server metrics

The following metrics are collected from the Kubernetes API server. These metrics help you monitor the health and performance of the control plane.

Metric Name	Dimensions	Description
`apiserver_storage_objects`	`ClusterName`	The number of objects stored in etcd by the API server.
`apiserver_request_total`	`ClusterName`	The total number of requests received by the API server.
`apiserver_request_duration_seconds`	`ClusterName`	The latency of API server requests, in seconds.

Send logs to CloudWatch Logs

To send logs from your containers to Amazon CloudWatch Logs, you can use Fluent Bit. For more information, see Fluent Bit.

> **Note**
>
> As of February 10 2025, AWS has deprecated support for FluentD as a log forwarder to CloudWatch Logs. We recommend that you use Fluent Bit, which is a lightweight and resource-efficient alternative. Existing FluentD deployments will continue to function. Migrate your logging pipeline to Fluent Bit to make sure continued support and optimal performance.
> Container Insights previously also supported using FluentD to send logs from your containers. FluentD has been deprecated and is now not supported for Container Insights. Use Fluent Bit instead.

Topics

- Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs
- (Optional) Set up Amazon EKS control plane logging
- (Optional) Enable the Use_Kubelet feature for large clusters

Set up Fluent Bit as a DaemonSet to send logs to CloudWatch Logs

The following sections help you deploy Fluent Bit to send logs from containers to CloudWatch Logs.

Topics

- Setting up Fluent Bit
- Multi-line log support
- (Optional) Reducing the log volume from Fluent Bit
- Troubleshooting
- Dashboard

Setting up Fluent Bit

To set up Fluent Bit to collect logs from your containers, you can follow the steps in Setup guide (AWS CLI) or you can follow the steps in this section.

With either method, the IAM role that is attached to the cluster nodes must have sufficient permissions. For more information about the permissions required to run an Amazon EKS cluster, see Amazon EKS IAM Policies, Roles, and Permissions in the *Amazon EKS User Guide*.

In the following steps, you set up Fluent Bit as a daemonSet to send logs to CloudWatch Logs. When you complete this step, Fluent Bit creates the following log groups if they don't already exist.

> **Important**
>
> If you already have FluentD configured in Container Insights and the FluentD DaemonSet is not running as expected (this can happen if you use the `containerd` runtime), you must uninstall it before installing Fluent Bit to prevent Fluent Bit from processing the FluentD error log messages. Otherwise, you must uninstall FluentD immediately after you have successfully installed Fluent Bit. Uninstalling Fluentd after installing Fluent Bit ensures continuity in logging during this migration process. Only one of Fluent Bit or FluentD is needed to send logs to CloudWatch Logs.

Log group name	Log source
/aws/containerinsights/ *Cluster_Name* /application	All log files in /var/log/containers
/aws/containerinsights/ *Cluster_Name* /host	Logs from /var/log/dmesg , /var/log/secure , and /var/log/messages
/aws/containerinsights/ *Cluster_Name* /dataplane	The logs in /var/log/journal for kubelet.service , kubeproxy.service , and docker.service .

To install Fluent Bit to send logs from containers to CloudWatch Logs

1. If you don't already have a namespace called amazon-cloudwatch, create one by entering the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
daemonset/container-insights-monitoring/cloudwatch-namespace.yaml
```

2. Run the following command to create a ConfigMap named cluster-info with the cluster name and the Region to send logs to. Replace *cluster-name* and *cluster-region* with your cluster's name and Region.

```
ClusterName=cluster-name
RegionName=cluster-region
FluentBitHttpPort='2020'
FluentBitReadFromHead='Off'
[[ ${FluentBitReadFromHead} = 'On' ]] && FluentBitReadFromTail='Off'||
 FluentBitReadFromTail='On'
[[ -z ${FluentBitHttpPort} ]] && FluentBitHttpServer='Off' ||
 FluentBitHttpServer='On'
kubectl create configmap fluent-bit-cluster-info \
--from-literal=cluster.name=${ClusterName} \
--from-literal=http.server=${FluentBitHttpServer} \
--from-literal=http.port=${FluentBitHttpPort} \
--from-literal=read.head=${FluentBitReadFromHead} \
--from-literal=read.tail=${FluentBitReadFromTail} \
```

```
--from-literal=logs.region=${RegionName} -n amazon-cloudwatch
```

In this command, the `FluentBitHttpServer` for monitoring plugin metrics is on by default. To turn it off, change the third line in the command to `FluentBitHttpPort=''` (empty string) in the command.

Also by default, Fluent Bit reads log files from the tail, and will capture only new logs after it is deployed. If you want the opposite, set `FluentBitReadFromHead='On'` and it will collect all logs in the file system.

3. Download and deploy the Fluent Bit daemonset to the cluster by running one of the following commands.

   - If you want the Fluent Bit optimized configuration for Linux computers, run this command.

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-
     bit.yaml
     ```

   - If you want the Fluent Bit optimized configuration for Windows computers, run this command.

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-
     windows.yaml
     ```

   - If you are using Linux computers and want the Fluent Bit configuration that is more similar to Fluentd, run this command.

     ```
     kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-
     cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/
     deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-
     compatible.yaml
     ```

> **Important**
>
> The Fluent Bit daemonset configuration by default sets the log level to INFO, which can result in higher CloudWatch Logs ingestion costs. If you want to reduce log ingestion volume and costs, you can change the log level to ERROR.
> For more information about how to reduce the log volume, see (Optional) Reducing the log volume from Fluent Bit

4. Validate the deployment by entering the following command. Each node should have one pod named **fluent-bit-***.

```
kubectl get pods -n amazon-cloudwatch
```

The above steps create the following resources in the cluster:

- A service account named `Fluent-Bit` in the `amazon-cloudwatch` namespace. This service account is used to run the Fluent Bit daemonSet. For more information, see Managing Service Accounts in the Kubernetes Reference.
- A cluster role named `Fluent-Bit-role` in the `amazon-cloudwatch` namespace. This cluster role grants `get`, `list`, and `watch` permissions on pod logs to the `Fluent-Bit` service account. For more information, see API Overview in the Kubernetes Reference.
- A ConfigMap named `Fluent-Bit-config` in the `amazon-cloudwatch` namespace. This ConfigMap contains the configuration to be used by Fluent Bit. For more information, see Configure a Pod to Use a ConfigMap in the Kubernetes Tasks documentation.

If you want to verify your Fluent Bit setup, follow these steps.

Verify the Fluent Bit setup

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Make sure that you're in the Region where you deployed Fluent Bit.
4. Check the list of log groups in the Region. You should see the following:
   - `/aws/containerinsights/`*`Cluster_Name`*`/application`

- `/aws/containerinsights/Cluster_Name/host`
- `/aws/containerinsights/Cluster_Name/dataplane`

5. Navigate to one of these log groups and check the **Last Event Time** for the log streams. If it is recent relative to when you deployed Fluent Bit, the setup is verified.

   There might be a slight delay in creating the `/dataplane` log group. This is normal as these log groups only get created when Fluent Bit starts sending logs for that log group.

Multi-line log support

For information on how to use Fluent Bit with multi-line logs, see the following sections of the Fluent Bit documentation:

- Multiline Parsing
- Multiline and Containers (v1.8)
- Multiline Core (v1.8)
- Always use multiline in the tail input

(Optional) Reducing the log volume from Fluent Bit

By default, we send Fluent Bit application logs and Kubernetes metadata to CloudWatch. If you want to reduce the volume of data being sent to CloudWatch, you can stop one or both of these data sources from being sent to CloudWatch. If you have followed the steps on this page to set up Fluent Bit, download the Kubernetes manifest YAML file from the kubectl `apply` command that you previously ran and modify it with your changes, which you can then re-apply to your cluster. Alternatively, if you are using the Amazon CloudWatch Observability EKS add-on or Helm chart, see (Optional) Additional configuration for information about managing the Fluent Bit configuration by using the add-on's advanced config or the Helm chart.

To stop Fluent Bit application logs, remove the following section from the `Fluent Bit configuration` file.

```
[INPUT]
  Name                tail
  Tag                 application.*
  Path                /var/log/containers/fluent-bit*
  Parser              docker
```

```
  DB                  /fluent-bit/state/flb_log.db
  Mem_Buf_Limit       5MB
  Skip_Long_Lines     On
  Refresh_Interval    10
```

To remove Kubernetes metadata from being appended to log events that are sent to CloudWatch, add the following filters to the `application-log.conf` section in the Fluent Bit configuration. Replace *<Metadata_1>* and the similar fields with with the actual metadata identifiers.

```
application-log.conf: |
    [FILTER]
  Name                nest
  Match               application.*
  Operation           lift
  Nested_under        kubernetes
  Add_prefix          Kube.

    [FILTER]
  Name                modify
  Match               application.*
  Remove              Kube.<Metadata_1>
  Remove              Kube.<Metadata_2>
  Remove              Kube.<Metadata_3>

    [FILTER]
  Name                nest
  Match               application.*
  Operation           nest
  Wildcard            Kube.*
  Nested_under        kubernetes
  Remove_prefix       Kube.
```

Troubleshooting

If you don't see these log groups and are looking in the correct Region, check the logs for the Fluent Bit daemonSet pods to look for the error.

Run the following command and make sure that the status is `Running`.

```
kubectl get pods -n amazon-cloudwatch
```

If the logs have errors related to IAM permissions, check the IAM role that is attached to the cluster nodes. For more information about the permissions required to run an Amazon EKS cluster, see Amazon EKS IAM Policies, Roles, and Permissions in the *Amazon EKS User Guide*.

If the pod status is `CreateContainerConfigError`, get the exact error by running the following command.

```
kubectl describe pod pod_name -n amazon-cloudwatch
```

Dashboard

You can create a dashboard to monitor metrics of each running plugin. You can see data for input and output bytes and for record processing rates as well as output errors and retry/failed rates. To view these metrics, you will need to install the CloudWatch agent with Prometheus metrics collection for Amazon EKS and Kubernetes clusters. For more information about how to set up the dashboard, see Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters.

> **Note**
>
> Before you can set up this dashboard, you must set up Container Insights for Prometheus metrics. For more information, see Container Insights Prometheus metrics monitoring.

To create a dashboard for the Fluent Bit Prometheus metrics

1. Create environment variables, replacing the values on the right in the following lines to match your deployment.

   ```
   DASHBOARD_NAME=your_cw_dashboard_name
   REGION_NAME=your_metric_region_such_as_us-west-1
   CLUSTER_NAME=your_kubernetes_cluster_name
   ```

2. Create the dashboard by running the following command.

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
   insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
   prometheus/sample_cloudwatch_dashboards/fluent-bit/cw_dashboard_fluent_bit.json \
   | sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
   | sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
   ```

```
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --
dashboard-body
```

(Optional) Set up Amazon EKS control plane logging

If you're using Amazon EKS, you can optionally enable Amazon EKS control plane logging, to provide audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs. For more information, see Amazon EKS Control Plane Logging.

(Optional) Enable the Use_Kubelet feature for large clusters

By default, the Use_Kubelet feature is disabled in the FluentBit Kubernetes plugin. Enabling this feature can reduce traffic to the API server and mitigate the issue of the API Server being a bottleneck. We recommend that you enable this feature for large clusters.

To enable Use_Kubelet, first add the nodes and nodes/proxy permissions to the clusterRole config.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluent-bit-role
rules:
  - nonResourceURLs:
- /metrics
    verbs:
- get
  - apiGroups: [""]
    resources:
- namespaces
- pods
- pods/logs
- nodes
- nodes/proxy
    verbs: ["get", "list", "watch"]
```

In the DaemonSet configuration, this feature needs host network access. The image version for `amazon/aws-for-fluent-bit` should 2.12.0 or later, or the fluent bit image version should be 1.7.2 or later.

```
apiVersion: apps/v1
kind: DaemonSet
```

```
metadata:
  name: fluent-bit
  namespace: amazon-cloudwatch
  labels:
    k8s-app: fluent-bit
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
k8s-app: fluent-bit
  template:
    metadata:
labels:
  k8s-app: fluent-bit
  version: v1
  kubernetes.io/cluster-service: "true"
    spec:
containers:
- name: fluent-bit
  image: amazon/aws-for-fluent-bit:2.19.0
  imagePullPolicy: Always
  env:
      - name: AWS_REGION
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: logs.region
      - name: CLUSTER_NAME
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: cluster.name
      - name: HTTP_SERVER
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: http.server
      - name: HTTP_PORT
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: http.port
      - name: READ_FROM_HEAD
```

```
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: read.head
      - name: READ_FROM_TAIL
        valueFrom:
          configMapKeyRef:
            name: fluent-bit-cluster-info
            key: read.tail
      - name: HOST_NAME
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      - name: HOSTNAME
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.name
      - name: CI_VERSION
        value: "k8s/1.3.8"
  resources:
      limits:
        memory: 200Mi
      requests:
        cpu: 500m
        memory: 100Mi
  volumeMounts:
  # Please don't change below read-only permissions
  - name: fluentbitstate
    mountPath: /var/fluent-bit/state
  - name: varlog
    mountPath: /var/log
    readOnly: true
  - name: varlibdockercontainers
    mountPath: /var/lib/docker/containers
    readOnly: true
  - name: fluent-bit-config
    mountPath: /fluent-bit/etc/
  - name: runlogjournal
    mountPath: /run/log/journal
    readOnly: true
  - name: dmesg
    mountPath: /var/log/dmesg
    readOnly: true
```

```
terminationGracePeriodSeconds: 10
hostNetwork: true
dnsPolicy: ClusterFirstWithHostNet
volumes:
- name: fluentbitstate
  hostPath:
    path: /var/fluent-bit/state
- name: varlog
  hostPath:
    path: /var/log
- name: varlibdockercontainers
  hostPath:
    path: /var/lib/docker/containers
- name: fluent-bit-config
  configMap:
    name: fluent-bit-config
- name: runlogjournal
  hostPath:
    path: /run/log/journal
- name: dmesg
  hostPath:
    path: /var/log/dmesg
serviceAccountName: fluent-bit
tolerations:
- key: node-role.kubernetes.io/master
  operator: Exists
  effect: NoSchedule
- operator: "Exists"
  effect: "NoExecute"
- operator: "Exists"
  effect: "NoSchedule"
```

The Kubernetes Plugin configuration should be similar to the following:

```
[FILTER]
  Name                kubernetes
  Match               application.*
  Kube_URL            https://kubernetes.default.svc:443
  Kube_Tag_Prefix     application.var.log.containers.
  Merge_Log           On
  Merge_Log_Key       log_processed
  K8S-Logging.Parser  On
  K8S-Logging.Exclude Off
```

```
  Labels              Off
  Annotations         Off
  Use_Kubelet         On
  Kubelet_Port        10250
  Buffer_Size         0
```

Compare Container Insights approaches

Two Container Insights approaches are available for Amazon EKS clusters, each with different capabilities, deployment models, and support levels. Both use the same `amazon-cloudwatch-observability` Amazon EKS add-on — the difference is the add-on version and configuration.

Use this page to understand the differences between the two approaches and determine which one is right for your environment. For detailed setup instructions, see OTel Container Insights (Recommended) or Enhanced Container Insights (Classic).

Feature comparison

The following table compares the two Container Insights approaches across key attributes.

Attribute	OTel Container Insights	Enhanced Container Insights (Classic)
Supported signals	Metrics, logs	Metrics, logs
Enhanced Observability	Yes	Yes
Maintenance status	Active development	Maintenance mode
Deployment complexity	Low	Low
Setup method	Amazon EKS add-on (OTel-based)	Amazon EKS add-on (managed)
Minimum Amazon EKS version	1.28	1.25
Minimum add-on version	v6.2.0	Prior to v6.2.0
Configuration model	Add-on configuration	Add-on configuration

Attribute	OTel Container Insights	Enhanced Container Insights (Classic)
Metric format	Prometheus/PromQL-native	CloudWatch-format (EMF)

Capabilities and limitations

Each approach has specific strengths and constraints. Review the following sections to understand what each approach provides and where it has limitations.

OTel Container Insights (Recommended)

OTel Container Insights provides the following capabilities:

- Full-signal observability through a single OpenTelemetry-based agent
- Active development with new features delivered regularly
- Simplified deployment as a managed Amazon EKS add-on
- PromQL-queryable metrics with up to 150 labels per metric

OTel Container Insights has the following limitations:

- Requires Amazon EKS 1.28 or later
- Requires add-on version v6.2.0 or later
- Newer offering with a smaller ecosystem of community examples

For setup instructions, see Quick start: OTel Container Insights on Amazon EKS.

Enhanced Container Insights (Classic)

Enhanced Container Insights (Classic) provides the following capabilities:

- Managed Amazon EKS add-on with automated lifecycle management
- Supports Enhanced Observability

Enhanced Container Insights (Classic) has the following limitations:

- In maintenance mode — receives security patches only

- Limited configuration flexibility
- Metrics use proprietary CloudWatch-format names, not compatible with standard PromQL dashboards

For setup instructions, see Setup guide (AWS CLI).

Decision guide

Use the following table to determine which approach is right for your situation.

Selection criteria	Recommended approach	Rationale
New Amazon EKS deployment	OTel Container Insights	Actively developed, full signal coverage, lowest complexity
Need full Enhanced Observability	OTel Container Insights	Only actively developed option
Already running Classic in production	Classic (short-term), OTel (migration target)	Continue for stability, then plan migration
Prefer managed add-on with minimal overhead	OTel Container Insights	Low configuration burden
Need PromQL-compatible metrics	OTel Container Insights	Preserves original Prometheus metric names

If you are currently running Enhanced Container Insights (Classic) and want to migrate, see Migration guides for step-by-step instructions.

Prerequisites summary

The following table lists the prerequisites for each approach.

Prerequisite	OTel Container Insights	Enhanced Container Insights (Classic)
Minimum Amazon EKS version	1.28	1.25
Minimum add-on version	v6.2.0	Prior to v6.2.0
IAM permissions	CloudWatchAgentServerPolicy	CloudWatchAgentServerPolicy
Required tools	kubectl, AWS CLI	kubectl, AWS CLI
OIDC provider required	Yes (for IRSA)	Yes (for IRSA)

Next steps

After you choose an approach, use the following links to get started:

- Quick start: OTel Container Insights on Amazon EKS — Set up OTel Container Insights with the fastest path.
- Setup guide (AWS CLI) — Set up Enhanced Container Insights (Classic).
- Migration guides — Migrate from Classic to OTel Container Insights.

Migration guides

This section provides step-by-step migration guides for moving from your current Container Insights setup to OTel Container Insights — the recommended, actively developed approach.

Why migrate?

OTel Container Insights is the recommended approach for Amazon EKS observability. Migrating provides the following benefits:

- **Active development** — OTel Container Insights receives new features, performance improvements, and expanded signal coverage on an ongoing basis.
- **Enhanced Observability** — Access detailed Kubernetes metrics at the pod, node, and cluster level with comprehensive log collection.

- **OpenTelemetry standard** — Built on the industry-standard observability framework, compatible with existing PromQL dashboards and community documentation.

What's covered

Each migration guide covers the following areas:

- **Parallel run** — Run both metrics streams simultaneously to validate data equivalence before switching over.
- **Breaking changes** — Removed features and changed defaults that might affect your workflows.
- **Migration steps** — A phased procedure for cutover with minimal monitoring gaps.
- **Verification** — Confirm that equivalent data flows through the new pipeline.
- **Rollback** — Restore your previous approach if needed.

Available migration guides

Choose the guide that matches your current Container Insights setup.

- Migrate from Enhanced Container Insights (Classic) to OTel Container Insights — Migrate from Enhanced Container Insights (Classic) to OTel Container Insights through an in-place add-on version update.

Migrate from Enhanced Container Insights (Classic) to OTel Container Insights

Both Enhanced Container Insights (Classic) and OTel Container Insights use the same `amazon-cloudwatch-observability` Amazon EKS add-on. The migration is an in-place add-on version update that you can complete in 15–30 minutes.

> **Warning**
>
> CloudWatch alarms configured with Classic metric names do not automatically work with OTel metrics. You must recreate alarms using PromQL-based alarm rules.

Prerequisites

Before you begin the migration, verify that you meet the following requirements.

- An Amazon EKS cluster running Kubernetes version 1.28 or later
- The `amazon-cloudwatch-observability` add-on installed and in `ACTIVE` status
- AWS CLI version 2.15.0 or later
- `kubectl` configured to communicate with your target cluster
- An IAM role with the `CloudWatchAgentServerPolicy` managed policy attached

Breaking changes

Review the following breaking changes before you begin the migration.

Removed features

The following features are not available in OTel Container Insights:

- **StatsD custom metrics** — Use the OTel StatsD receiver instead.
- **collectd plugin** — Migrate to OTel-native instrumentation.

Changed defaults

The following table shows default values that change between Classic and OTel Container Insights.

Setting	Classic default	OTel CI default
Collection interval	60 seconds	60 seconds
Enhanced Observability	Enabled	Enabled
Agent CPU request	200m	100m
Agent memory request	200Mi	128Mi

Migration steps

This migration uses a phased approach to minimize monitoring gaps. You run both Classic and OTel metrics streams in parallel, validate the data, and then disable the Classic stream.

Phase 1: Confirm current state (Classic only)

Before you make changes, confirm your current add-on state and record the version for rollback.

To confirm your current add-on configuration

1. Run the following command to check the current add-on status.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability
```

2. Verify that the `status` field is `ACTIVE`.
3. Record the `addonVersion` value. You need this value if you must roll back.

Phase 2: Enable dual publishing (Classic + OTel)

Enable OTel Container Insights alongside Enhanced Container Insights (Classic). This publishes both metric streams simultaneously so that you can validate data equivalence.

> **Important**
>
> Both metric streams incur charges during this phase. We recommend keeping the dual publishing window as short as possible to minimize costs.

To enable dual publishing

1. Run the following command to update the add-on with both streams enabled.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --addon-version latest-version \
  --configuration-values '{"containerInsights":
{"enabled":true},"otelContainerInsights":{"enabled":true}}' \
  --resolve-conflicts OVERWRITE
```

2. Wait for the add-on status to return to `ACTIVE`.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status"
```

3. Verify that OTel metrics appear in CloudWatch by querying a known metric with PromQL.

Phase 3: Recreate alarms and update dashboards

Build OTel replacements for your existing Classic alarms and dashboards. Keep Classic alarms active as a safety net during this phase.

To recreate alarms for OTel metrics

1. Identify all CloudWatch alarms that reference Classic Container Insights metric names.
2. Create equivalent alarms using PromQL-based metric math expressions that reference OTel metric names.
3. Verify that the new alarms enter OK state and produce equivalent threshold evaluations.
4. Update any CloudWatch dashboards to include widgets based on OTel metrics alongside the existing Classic widgets.

Phase 4: Disable Classic (switch to OTel only)

After you validate that OTel metrics, alarms, and dashboards work correctly, disable the Classic stream.

To disable Classic and keep OTel only

1. Run the following command to disable Classic metrics publishing.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --configuration-values '{"containerInsights":
{"enabled":false},"otelContainerInsights":{"enabled":true}}' \
  --resolve-conflicts OVERWRITE
```

2. Wait for the add-on status to return to `ACTIVE`.
3. Remove the Classic alarms that you replaced in Phase 3.

Verification

After you complete the migration, verify that your observability stack is working correctly.

- **Check metrics in Query Studio** — Use PromQL queries to confirm that metrics flow from the OTel pipeline.
- **Compare with baseline values** — Verify that metric values are consistent with the Classic metrics you recorded during Phase 2.
- **Verify Container Insights dashboards** — Confirm that the Container Insights console displays data for your cluster.
- **Verify log delivery** — Check that container logs continue to appear in CloudWatch Logs.
- **Verify alarms** — Confirm that no alarms are in `INSUFFICIENT_DATA` state.

Rollback

If you encounter issues after migration, you can restore the previous Classic configuration.

To roll back to Enhanced Container Insights (Classic)

1. Identify the previous add-on version that you recorded in Phase 1.
2. Run the following command to downgrade the add-on.

```
aws eks update-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --addon-version previous-version \
  --resolve-conflicts OVERWRITE
```

3. Wait for the add-on status to return to `ACTIVE`.

```
aws eks describe-addon \
  --cluster-name cluster-name \
  --addon-name amazon-cloudwatch-observability \
  --query "addon.status"
```

4. Re-apply any custom configuration values that you used with the previous version.

Troubleshooting Container Insights on Amazon EKS

This section covers common issues that you might encounter when you set up or operate Container Insights on Amazon EKS. Use the following tables and diagnostic commands to identify and resolve problems regardless of whether you use the OTel or Classic approach.

For approach-specific setup guidance, see Quick start: OTel Container Insights on Amazon EKS or Setup guide (AWS CLI). To compare approaches, see Compare Container Insights approaches.

Metrics not appearing in CloudWatch

If you don't see metrics in the `ContainerInsights` namespace, use the following table to identify the cause.

Symptom	Cause	Resolution
No metrics in the `ContainerInsights` namespace	IAM role lacks `cloudwatch:PutMetricData` permission	Attach the `CloudWatchAgentServerPolicy` managed policy to the agent IAM role.
Metrics appear for some nodes but not others	Agent DaemonSet not scheduled on all nodes because of taints	Add tolerations to the agent DaemonSet to allow scheduling on tainted nodes.
Metrics stop appearing	Agent pod is OOMKilled or restarting	Increase the memory limits in the agent pod resource specification.
Metrics are stale or zero	Network connectivity is blocked	Check VPC security groups and verify that a CloudWatch VPC endpoint exists.
Enhanced metrics are missing	Agent not configured for Enhanced Observability	Set `enhancedObservability: true` in the agent configuration.

Agent pods not starting

If agent pods fail to start or remain in a non-running state, use the following table to diagnose the issue.

Symptom	Cause	Resolution
`ImagePullBackOff`	Amazon ECR is unreachable or the image tag is incorrect	Verify the image URI and confirm that your nodes can access Amazon ECR.

Symptom	Cause	Resolution
`Pending`	Insufficient CPU or memory on the node	Scale the node group or reduce resource requests in the agent pod specification.
`CrashLoopBackOff`	Invalid configuration or missing volume mount	Check pod logs for configuration errors by running `kubectl logs` on the affected pod.
`FailedScheduling`	Node affinity or taints prevent scheduling	Review the `nodeSelector` and tolerations in the DaemonSet spec.
Exit code 1	Service account lacks IRSA annotation	Verify that the service account has the `eks.amazonaws.com/role-arn` annotation.

Add-on installation failures

If the `amazon-cloudwatch-observability` add-on fails to install or reports an unhealthy status, use the following table to troubleshoot.

Symptom	Cause	Resolution
`CREATE_FAILED`	Conflicting resources from a previous installation	Delete conflicting resources and use `--resolve-conflicts OVERWRITE` when you create the add-on.
OIDC provider not found	No IAM OIDC identity provider exists for the cluster	Create the provider by running `eksctl utils associate-iam-oidc-provider .`
Version conflict	Add-on version is incompatible with the Kubernetes version	List compatible versions by running `aws eks describe-addon-versions .`

Symptom	Cause	Resolution
DEGRADED status	Health checks are failing because of missing permissions	Check pod logs and verify that the IRSA role has the required policies attached.

Log delivery issues

If container logs don't appear in Amazon CloudWatch Logs, use the following table to identify the cause.

Symptom	Cause	Resolution
Log group doesn't exist	Missing `logs:CreateLogGroup` permission	Add Amazon CloudWatch Logs permissions to the agent IAM role.
Log group exists but is empty	Agent not configured for logs, or Region mismatch	Verify that the agent configuration includes log collection and that the Region matches your cluster Region.
Logs are delayed more than 5 minutes	Flush interval is too high or the node is under heavy load	Reduce the `force_flush_interval` value in the agent configuration.
Performance logs are missing	Agent is configured for application logs only	Verify that the Container Insights performance log section is present in the agent configuration.

Migration-specific issues

If you experience issues while migrating between Container Insights approaches, use the following table. For the full migration workflow, see Migration guides.

Symptom	Cause	Resolution
Duplicate metrics during parallel run	Both approaches are publishing metrics simultaneously	This behavior is expected during a parallel run. Disable the legacy

Symptom	Cause	Resolution
		approach after you validate the new approach.
Different metric values between approaches	Different calculation methods	Small differences (less than 5%) are expected. Large differences indicate a configuration mismatch between approaches.
Rollback fails	Custom configuration was not reapplied	Re-apply your complete configuration values when you roll back.
Alarms fire during migration	Metric gaps during the switchover period	Temporarily set the missing data treatment to `notBreaching` on affected alarms.

OTel Container Insights issues

The following issues are specific to the OTel Container Insights approach. For general setup guidance, see Quick start: OTel Container Insights on Amazon EKS.

Symptom	Cause	Resolution
403 Forbidden exporter error	IAM role is missing CloudWatch permissions	Verify that the `CloudWatchAgentServerPolicy` is attached to the agent role.
Connection refused on metrics endpoint	Collector cannot reach the kubelet	Verify that `hostNetwork: true` is set in the pod spec, or confirm that the service account has the required permissions.
High memory usage	Batch processor queue is too large	Reduce the `batch/timeout` and `batch/send_batch_size` values in the collector configuration.

Symptom	Cause	Resolution
Custom metrics not appearing	Receiver not configured for the application endpoint	Add a Prometheus receiver that targets your application metrics port in the collector configuration.

General diagnostic commands

Use the following commands to gather information about your Container Insights deployment.

To check agent pod status, run the following command.

```
kubectl get pods -n amazon-cloudwatch
```

To view agent pod logs, run the following command.

```
kubectl logs -n amazon-cloudwatch -l app.kubernetes.io/name=cloudwatch-agent --tail=50
```

To check the agent DaemonSet status, run the following command.

```
kubectl get daemonset -n amazon-cloudwatch
```

To verify the IAM role on a service account, run the following command.

```
kubectl get serviceaccount -n amazon-cloudwatch -o yaml
```

To check the cluster add-on status, run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws eks describe-addon --cluster-name cluster-name --addon-name amazon-cloudwatch-
observability
```

To list Container Insights log groups, run the following command. Replace *cluster-name* with the name of your Amazon EKS cluster.

```
aws logs describe-log-groups --log-group-name-prefix "/aws/containerinsights/cluster-
name"
```

Related resources

For more information about setting up and operating Container Insights on Amazon EKS, see the following topics.

- Quick start: OTel Container Insights on Amazon EKS – Set up OTel Container Insights
- Setup guide (AWS CLI) – Set up Classic Container Insights
- Migration guides – Migrate between approaches
- Compare Container Insights approaches – Compare Container Insights approaches

Amazon ECS

You can use one or both of the following options to enable Container Insights on Amazon ECS clusters:

- Use the AWS Management Console or the AWS CLI to start collecting cluster-level, task-level, and service-level metrics.
- Deploy the CloudWatch agent as a daemon service to start collecting of instance-level metrics on clusters that are hosted on Amazon EC2 instances.

Topics

- Setting up Container Insights on Amazon ECS
- Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry
- Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS
- Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters
- Set up FireLens to send logs to CloudWatch Logs

Setting up Container Insights on Amazon ECS

You can set up Container Insights with enhanced observability or Container Insights on new and existing Amazon ECS clusters using either the Amazon ECS console or the AWS CLI. Container Insights collects metrics at the cluster, task, and service levels. Container Insights with enhanced observability provides additional dimensions and metrics, allowing you to deep dive down to container level visibility.

If you're using Amazon ECS on an Amazon EC2 instance, launch that instance using an AMI that includes Amazon ECS agent version 1.29 or later. For information about updating your agent version, see Updating the Amazon ECS Container Agent.

> **Note**
>
> If the customer managed AWS KMS key that you use for your Amazon ECS Container Insights metrics is not already configured to work with CloudWatch, you must update the key policy to allow for encrypted logs in CloudWatch Logs. You must also associate your own AWS KMS key with the log group in `/aws/ecs/containerinsights/ClusterName/performance`. For more information, see Encrypt log data in CloudWatch Logs using AWS Key Management Service.

We recommend that you use Container Insights with enhanced observability instead of Container Insights as it provides detailed visibility in your container environment, reducing the mean time to resolution.

Set up Container Insights with enhanced observability

You can turn on Container Insights with enhanced observability using the Amazon ECS console or AWS CLI.

AWS CLI

Use the following command to turn on Container Insights with enhanced observability.

Set the `containerInsights` account setting to `enhanced`

```
aws ecs put-account-setting --name containerInsights --value enhanced
```

Example output

```
{
  "setting": {
      "name": "containerInsights",
      "value": "enhanced",
      "principalArn": "arn:aws:iam::123456789012:johndoe",
       "type": user
  }
```

```
}
```

> **Note**
>
> By default, the `put-account-setting` applies only to the currently authenticated user. To enable the setting account-wide for all users and roles, use the root user as in the following example.
>
> ```
> aws ecs put-account-setting --name containerInsights --value enhanced --
> principal-arn arn:aws:iam::accountID:root
> ```

After you set this account setting, all new clusters automatically use Container Insights with enhanced observability. Use the `update-cluster-settings` command to add Container Insights with enhanced observability to existing cluster, or to upgrade clusters that currently use Container Insights to Container Insights with enhanced observability.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enhanced
```

Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. To use Container Insights with enhanced observability, choose **Container Insights with enhanced observability**.
6. Choose **Save changes**.
7. On the confirmation screen, choose **Confirm** to save the selection.

After you set this, all new clusters automatically use Container Insights with enhanced observability. You can add Container Insights with enhanced observability to existing clusters, or update clusters that currently use Container Insights to Container Insights with enhanced observability. For more information, see Updating an Amazon ECS cluster in the *Amazon Elastic Container Service Developer Guide*.

Set up Container Insights

You can turn on Container Insights using the Amazon ECS console or AWS CLI.

AWS CLI

To use Container Insights, set the `container Insights` account setting to `enabled`. Use the following command to turn on Container Insights.

```
aws ecs put-account-setting --name containerInsights --value enabled
```

Example output

```
{
  "setting": {
      "name": "container Insights",
      "value": "enabled",
      "principalArn": "arn:aws:iam::123456789012:johndoe",
       "type": user
  }
}
```

When you set the `container Insights` account setting to `enabled`, all new clusters have Container Insights enabled by default. Use the `update-cluster-settings` command to add Container Insights to an existing cluster.

```
aws ecs update-cluster-settings --cluster cluster-name --settings
 name=containerInsights,value=enabled
```

Amazon ECS console

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation bar at the top, select the Region for which to view your account settings.
3. In the navigation page, choose **Account Settings**.
4. Choose **Update**.
5. To use Container Insights, choose **Container Insights**.
6. Choose **Save changes**.
7. On the confirmation screen, choose **Confirm** to save the selection.

After you set this, all new clusters automatically use Container Insights. Update existing clusters to add Container Insights. For more information, see Updating an Amazon ECS cluster in the *Amazon Elastic Container Service Developer Guide*.

Setting up Container Insights on Amazon ECS using AWS Distro for OpenTelemetry

Use this section if you want to use AWS Distro for OpenTelemetry to set up CloudWatch Container Insights on an Amazon ECS cluster. For more information about AWS Distro for Open Telemetry, see AWS Distro for OpenTelemetry.

These steps assume that you already have a cluster running Amazon ECS. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service.

Step 1: Create a task role

The first step is creating a task role in the cluster that the AWS OpenTelemetry Collector will use.

To create a task role for AWS Distro for OpenTelemetry

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab and copy in the following policy:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
```

```
                "ssm:GetParameters"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Choose **Review policy**.
5. For name, enter **`AWSDistroOpenTelemetryPolicy`**, and then choose **Create policy**.
6. In the left navigation pane, choose **Roles** and then choose **Create role**.
7. In the list of services, choose **Elastic Container Service**.
8. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
9. In the list of policies, search for **AWSDistroOpenTelemetryPolicy**.
10. Select the check box next to **AWSDistroOpenTelemetryPolicy**.
11. Choose **Next: Tags** and then choose **Next: Review.**
12. For **Role name** enter **`AWSOpenTelemetryTaskRole`** and then choose **Create role**.

Step 2: Create a task execution role

The next step is creating a task execution role for the AWS OpenTelemetry Collector.

To create a task execution role for AWS Distro for OpenTelemetry

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles** and then choose **Create role**.
3. In the list of services, choose **Elastic Container Service**.
4. Lower on the page, choose **Elastic Container Service Task** and then choose **Next: Permissions**.
5. In the list of policies, search for **AmazonECSTaskExecutionRolePolicy** and then select the check box next to **AmazonECSTaskExecutionRolePolicy**.
6. In the list of policies, search for **CloudWatchLogsFullAccess** and then select the check box next to **CloudWatchLogsFullAccess**.
7. In the list of policies, search for **AmazonSSMReadOnlyAccess** and then select the check box next to **AmazonSSMReadOnlyAccess**.
8. Choose **Next: Tags** and then choose **Next: Review.**

9. For **Role name** enter **`AWSOpenTelemetryTaskExecutionRole`** and then choose **Create role**.

Step 3: Create a task definition

The next step is creating a task definition.

To create a task definition for AWS Distro for OpenTelemetry

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the navigation pane, choose **Task definitions**
3. Choose **Create new task definition**, **Create new task definition**.
4. For **Task definition family**, specify a unique name for the task definition.
5. Configure your containers, and then choose **Next**.
6. Under **Metrics and logging**, select **Use metric collection**.
7. Choose **Next**.
8. Choose **Create**.

For more information about using the AWS OpenTelemetry collector with Amazon ECS, see Setting up AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service.

Step 4: Run the task

The final step is running the task that you've created.

To run the task for AWS Distro for OpenTelemetry

1. Open the console at https://console.aws.amazon.com/ecs/v2.
2. In the left navigation pane, choose **Task Definitions** and then select the task that you just created.
3. Choose **Actions**, **Deploy**, **Run task**.
4. Choose **Deploy**, **Run task**.
5. In the **Compute options** section, from **Existing cluster**, choose the cluster.
6. Choose **Create**.
7. Next, you can check for the new metrics in the CloudWatch console.
8. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

9. In the left navigation pane, choose **Metrics**.

   You should see a **ECS/ContainerInsights** namespace. Choose that namespace and you should see eight metrics.

Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS

To deploy the CloudWatch agent to collect instance-level metrics from Amazon ECS clusters that are hosted on EC2 instance, use a quick start setup with a default configuration, or install the agent manually to be able to customize it.

Both methods require that you already have at least one Amazon ECS cluster deployed with an EC2 launch type and that the CloudWatch agent continer has access to the Amazon EC2 Instance Metadata Service (IMDS). For more information about IMDS, see Instance metadata and user data.

These methods also assume that you have the AWS CLI installed. Additionally, to run the commands in the following procedures, you must be logged on to an account or role that has the **IAMFullAccess** and **AmazonECS_FullAccess** policies.

> **Important**
>
> When defining the CloudWatch Agent container in your task definition, set `essential: false`. This prevents the entire Amazon ECS service from stopping if the CloudWatch Agent container fails. Other critical application containers will continue running even if the agent is temporarily unavailable.

Topics

- Quick setup using CloudFormation
- Manual and custom setup

Quick setup using CloudFormation

To use the quick setup, enter the following command to use CloudFormation to install the agent. Replace *`cluster-name`* and *`cluster-region`* with the name and Region of your Amazon ECS cluster.

This command creates the IAM roles **CWAgentECSTaskRole** and **CWAgentECSExecutionRole**. If these roles already exist in your account, use `ParameterKey=CreateIAMRoles,ParameterValue=False` instead of `ParameterKey=CreateIAMRoles,ParameterValue=True` when you enter the command. Otherwise, the command will fail.

```
ClusterName=cluster-name
Region=cluster-region
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
  --template-body file://cwagent-ecs-instance-metric-cfn.json \
  --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
               ParameterKey=CreateIAMRoles,ParameterValue=True \
  --capabilities CAPABILITY_NAMED_IAM \
  --region ${Region}
```

(Alternative) Using your own IAM roles

If you want to use your own custom ECS task role and ECS task execution role instead of the **CWAgentECSTaskRole** and **CWAgentECSExecutionRole** roles, first make sure that the role to be used as the ECS task role has **CloudWatchAgentServerPolicy** attached. Also, make sure that the role to be used as the ECS task execution role has both the **CloudWatchAgentServerPolicy** and **AmazonECSTaskExecutionRolePolicy** policies attached. Then enter the following command. In the command, replace *task-role-arn* with the ARN of your custom ECS task role, and replace *execution-role-arn* with the ARN of your custom ECS task execution role.

```
ClusterName=cluster-name
Region=cluster-region
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cloudformation-quickstart/cwagent-ecs-instance-metric-cfn.json
aws cloudformation create-stack --stack-name CWAgentECS-${ClusterName}-${Region} \
  --template-body file://cwagent-ecs-instance-metric-cfn.json \
  --parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
               ParameterKey=TaskRoleArn,ParameterValue=${TaskRoleArn} \
               ParameterKey=ExecutionRoleArn,ParameterValue=${ExecutionRoleArn} \
  --capabilities CAPABILITY_NAMED_IAM \
```

```
  --region ${Region}
```

Troubleshooting the quick setup

To check the status of the CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stacks --stack-name CWAgentECS-$ClusterName-$Region --
region $Region
```

If you see the `StackStatus` is other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name CWAgentECS-$ClusterName-$Region
 --region $Region
```

To check the status of the `cwagent` daemon service, enter the following command. In the output, you should see that the `runningCount` is equal to the `desiredCount` in the `deployment` section. If it isn't equal, check the `failures` section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services cwagent-daemon-service --cluster $ClusterName --
region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/ecs/ecs-cwagent-daemon-service** log group.

Deleting the CloudFormation stack for the CloudWatch agent

If you need to delete the CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation delete-stack --stack-name CWAgentECS-${ClusterName}-${Region} --
region ${Region}
```

Manual and custom setup

Follow the steps in this section to manually deploy the CloudWatch agent to collect instance-level metrics from your Amazon ECS clusters that are hosted on EC2 instances.

Necessary IAM roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see IAM roles for Tasks and Amazon ECS Task Execution Role.

- An *ECS task role*, which is used by the CloudWatch agent to publish metrics. If this role already exists, you must make sure it has the `CloudWatchAgentServerPolicy` policy attached.
- An *ECS task execution role*, which is used by Amazon ECS agent to launch the CloudWatch agent. If this role already exists, you must make sure it has the `AmazonECSTaskExecutionRolePolicy` and `CloudWatchAgentServerPolicy` policies attached.

If you do not already have these roles, you can use the following commands to create them and attach the necessary policies. This first command creates the ECS task role.

```
aws iam create-role --role-name CWAgentECSTaskRole \
  --assume-role-policy-document "{\"Version\": \"2012-10-17\",       \"Statement
\": [{\"Sid\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-
tasks.amazonaws.com\"},\"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as "TaskRoleArn". You'll need to use it later when you create the task definition. Then enter the following command to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
  --role-name CWAgentECSTaskRole
```

This next command creates the ECS task execution role.

```
aws iam create-role --role-name CWAgentECSExecutionRole \
  --assume-role-policy-document "{\"Version\": \"2012-10-17\",       \"Statement
\": [{\"Sid\": \"\",\"Effect\": \"Allow\",\"Principal\": {\"Service\": \"ecs-
tasks.amazonaws.com\"},\"Action\": \"sts:AssumeRole\"}]}"
```

After you enter the previous command, note the value of `Arn` from the command output as "ExecutionRoleArn". You'll need to use it later when you create the task definition. Then enter the following commands to attach the necessary policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy \
  --role-name CWAgentECSExecutionRole

aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy \
  --role-name CWAgentECSExecutionRole
```

Create the task definition and launch the daemon service

Create a task definition and use it to launch the CloudWatch agent as a daemon service. To create the task definition, enter the following command. In the first lines, replace the placeholders with the actual values for your deployment. *`logs-region`* is the Region where CloudWatch Logs is located, and *`cluster-region`* is the Region where your cluster is located. *`task-role-arn`* is the Arn of the ECS task role that you are using, and *`execution-role-arn`* is the Arn of the ECS task execution role.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-
ecs-instance-metric/cwagent-ecs-instance-metric.json \
  | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
  | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-json
```

Then run the following command to launch the daemon service. Replace *`cluster-name`* and *`cluster-region`* with the name and Region of your Amazon ECS cluster.

> **Important**
>
> Remove all capacity provider strategies before you run this command. Otherwise, the command won't work.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
  --cluster ${ClusterName} \
  --service-name cwagent-daemon-service \
  --task-definition ecs-cwagent-daemon-service \
  --scheduling-strategy DAEMON \
  --region ${Region}
```

If you see this error message, `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`, you have already created a daemon service named `cwagent-daemon-service`. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs delete-service \
  --cluster ${ClusterName} \
  --service cwagent-daemon-service \
  --region ${Region} \
  --force
```

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the CloudWatch agent in your Amazon ECS clusters that are hosted on EC2 instances. These options are as follows:

- `metrics_collection_interval` – How often in seconds that the CloudWatch agent collects metrics. The default is 60. The range is 1–172,000.
- `endpoint_override` – (Optional) Specifies a different endpoint to send logs to. You might want to do this if you're publishing from a cluster in a VPC and you want the logs data to go to a VPC endpoint.

  The value of `endpoint_override` must be a string that is a URL.
- `force_flush_interval` – Specifies in seconds the maximum amount of time that logs remain in the memory buffer before being sent to the server. No matter the setting for this field, if the size of the logs in the buffer reaches 1 MB, the logs are immediately sent to the server. The default value is 5 seconds.

- `region` – By default, the agent publishes metrics to the same Region where the Amazon ECS container instance is located. To override this, you can specify a different Region here. For example, `"region" : "us-east-1"`

The following is an example of a customized configuration:

```
{
  "agent": {
      "region": "us-east-1"
  },
  "logs": {
      "metrics_collected": {
          "ecs": {
              "metrics_collection_interval": 30
          }
      },
      "force_flush_interval": 5
  }
}
```

To customize your CloudWatch agent configuration in your Amazon ECS containers

1. Make sure that the **AmazonSSMReadOnlyAccess** policy is attached to your Amazon ECS Task Execution role. You can enter the following command to do so. This example assumes that your Amazon ECS Task Execution role is CWAgentECSExecutionRole. If you are using a different role, substitute that role name in the following command.

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonSSMReadOnlyAccess \
         --role-name CWAgentECSExecutionRole
   ```

2. Create the customized configuration file similar to the preceding example. Name this file `/tmp/ecs-cwagent-daemon-config.json`.
3. Run the following command to put this configuration into the Parameter Store. Replace *`cluster-region`* with the Region of your Amazon ECS cluster. To run this command, you must be logged on to a user or role that has the **AmazonSSMFullAccess** policy.

   ```
   Region=cluster-region
   aws ssm put-parameter \
     --name "ecs-cwagent-daemon-service" \
   ```

```
  --type "String" \
  --value "`cat /tmp/ecs-cwagent-daemon-config.json`" \
  --region $Region
```

4. Download the task definition file to a local file, such as `/tmp/cwagent-ecs-instance-metric.json`

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/daemon-service/cwagent-ecs-instance-metric/cwagent-ecs-instance-metric.json -o /tmp/cwagent-ecs-instance-metric.json
```

5. Modify the task definition file. Remove the following section:

```
"environment": [
              {
                  "name": "USE_DEFAULT_CONFIG",
                  "value": "True"
              }
          ],
```

Replace that section with the following:

```
"secrets": [
              {
                  "name": "CW_CONFIG_CONTENT",
                  "valueFrom": "ecs-cwagent-daemon-service"
              }
          ],
```

6. Restart the agent as a daemon service by following these steps:

   a. Run the following command.

```
TaskRoleArn=task-role-arn
ExecutionRoleArn=execution-role-arn
AWSLogsRegion=logs-region
Region=cluster-region
cat /tmp/cwagent-ecs-instance-metric.json \
  | sed "s|{{task-role-arn}}|${TaskRoleArn}|;s|{{execution-role-arn}}|
${ExecutionRoleArn}|;s|{{awslogs-region}}|${AWSLogsRegion}|" \
```

```
  | xargs -0 aws ecs register-task-definition --region ${Region} --cli-input-
json
```

b. Run the following command to launch the daemon service. Replace *cluster-name* and *cluster-region* with the name and Region of your Amazon ECS cluster.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs create-service \
  --cluster ${ClusterName} \
  --service-name cwagent-daemon-service \
  --task-definition ecs-cwagent-daemon-service \
  --scheduling-strategy DAEMON \
  --region ${Region}
```

If you see this error message, `An error occurred (InvalidParameterException) when calling the CreateService operation: Creation of service was not idempotent`, you have already created a daemon service named `cwagent-daemon-service`. You must delete that service first, using the following command as an example.

```
ClusterName=cluster-name
Region=Region
aws ecs delete-service \
  --cluster ${ClusterName} \
  --service cwagent-daemon-service \
  --region ${Region} \
  --force
```

Deploying the AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on Amazon ECS clusters

Use the steps in this section to use AWS Distro for OpenTelemetry to collect EC2 instance-level metrics on an Amazon ECS cluster. For more information about the AWS Distro for OpenTelemetry, see AWS Distro for OpenTelemetry.

These steps assume that you already have a cluster running Amazon ECS. This cluster must be deployed with the EC2 launch type. For more information about using AWS Distro for Open Telemetry with Amazon ECS and setting up an Amazon ECS cluster for this purpose, see Setting up

AWS Distro for OpenTelemetry Collector in Amazon Elastic Container Service for ECS EC2 instance level metrics.

Topics

- Quick setup using CloudFormation
- Manual and custom setup

Quick setup using CloudFormation

Download the CloudFormation template file for installing the AWS Distro for OpenTelemetry collector for Amazon ECS on EC2. Run the following curl command.

```
curl -O https://raw.githubusercontent.com/aws-observability/aws-otel-collector/main/
deployment-template/ecs/aws-otel-ec2-instance-metrics-daemon-deployment-cfn.yaml
```

After you download the template file, open it and replace *PATH_TO_CloudFormation_TEMPLATE* with the path where you saved the template file. Then export the following parameters and run the CloudFormation command, as shown in the following command.

- **Cluster_Name**– The Amazon ECS cluster name
- **AWS_Region**– The Region where the data will be sent
- **PATH_TO_CloudFormation_TEMPLATE**– The path where you saved the CloudFormation template file.
- **command**– To enable the AWS Distro for OpenTelemetry collector to collect the instance-level metrics for Amazon ECS on Amazon EC2, you must specify `--config=/etc/ecs/otel-instance-metrics-config.yaml` for this parameter.

```
ClusterName=Cluster_Name
Region=AWS_Region
command=--config=/etc/ecs/otel-instance-metrics-config.yaml
aws cloudformation create-stack --stack-name AOCECS-${ClusterName}-${Region} \
--template-body file://PATH_TO_CloudFormation_TEMPLATE \
--parameters ParameterKey=ClusterName,ParameterValue=${ClusterName} \
ParameterKey=CreateIAMRoles,ParameterValue=True \
ParameterKey=command,ParameterValue=${command} \
--capabilities CAPABILITY_NAMED_IAM \
--region ${Region}
```

After running this command, use the Amazon ECS console to see if the task is running.

Troubleshooting the quick setup

To check the status of the CloudFormation stack, enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack --stack-name AOCECS-$ClusterName-$Region --region
 $Region
```

If the value of `StackStatus` is anything other than `CREATE_COMPLETE` or `CREATE_IN_PROGRESS`, check the stack events to find the error. Enter the following command.

```
ClusterName=cluster-name
Region=cluster-region
aws cloudformation describe-stack-events --stack-name AOCECS-$ClusterName-$Region --
region $Region
```

To check the status of the `AOCECS` daemon service, enter the following command. In the output, you should see that `runningCount` is equal to the `desiredCount` in the deployment section. If it isn't equal, check the failures section in the output.

```
ClusterName=cluster-name
Region=cluster-region
aws ecs describe-services --services AOCECS-daemon-service --cluster $ClusterName --
region $Region
```

You can also use the CloudWatch Logs console to check the agent log. Look for the **/aws/ecs/containerinsights/{ClusterName}/performance** log group.

Manual and custom setup

Follow the steps in this section to manually deploy the AWS Distro for OpenTelemetry to collect instance-level metrics from your Amazon ECS clusters that are hosted on Amazon EC2 instances.

Step 1: Necessary roles and policies

Two IAM roles are required. You must create them if they don't already exist. For more information about these roles, see Create IAM policy and Create IAM role.

Step 2: Create the task definition

Create a task definition and use it to launch the AWS Distro for OpenTelemetry as a daemon service.

To use the task definition template to create the task definition, follow the instructions in Create ECS EC2 Task Definition for EC2 instance with AWS OTel Collector.

To use the Amazon ECS console to create the task definition, follow the instructions in Install AWS OTel Collector by creating Task Definition through AWS console for Amazon ECS EC2 instance metrics.

Step 3: Launch the daemon service

To launch the AWS Distro for OpenTelemetry as a daemon service, follow the instructions in Run your task on the Amazon Elastic Container Service (Amazon ECS) using daemon service.

(Optional) Advanced configuration

Optionally, you can use SSM to specify other configuration options for the AWS Distro for OpenTelemetry in your Amazon ECS clusters that are hosted on Amazon EC2 instances. For more information, about creating a configuration file, see Custom OpenTelemetry Configuration. For more information about the options that you can use in the configuration file, see AWS Container Insights Receiver.

Set up FireLens to send logs to CloudWatch Logs

FireLens for Amazon ECS enables you to use task definition parameters to route logs to Amazon CloudWatch Logs for log storage and analytics. FireLens works with Fluent Bit and Fluentd. We provide an AWS for Fluent Bit image, or you can use your own Fluent Bit or Fluentd image. Creating Amazon ECS task definitions with a FireLens configuration is supported using the AWS SDKs, AWS CLI, and AWS Management Console. For more information about CloudWatch Logs, see What is CloudWatch Logs?.

There are key considerations when using FireLens for Amazon ECS. For more information, see Considerations.

To find the AWS for Fluent Bit images, see Using the AWS for Fluent Bit image.

To create a task definition that uses a FireLens configuration, see Creating a task definition that uses a FireLens configuration.

Example

The following task definition example demonstrates how to specify a log configuration that forwards logs to a CloudWatch Logs log group. For more information, see What Is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*.

In the log configuration options, specify the log group name and the Region it exists in. To have Fluent Bit create the log group on your behalf, specify `"auto_create_group":"true"`. You can also specify the task ID as the log stream prefix, which assists in filtering. For more information, see Fluent Bit Plugin for CloudWatch Logs.

```
{
 "family": "firelens-example-cloudwatch",
 "taskRoleArn": "arn:aws:iam::123456789012:role/ecs_task_iam_role",
 "containerDefinitions": [
  {
   "essential": true,
   "image": "906394416424.dkr.ecr.us-west-2.amazonaws.com/aws-for-fluent-bit:latest",
   "name": "log_router",
   "firelensConfiguration": {
    "type": "fluentbit"
   },
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "firelens-container",
     "awslogs-region": "us-west-2",
     "awslogs-create-group": "true",
     "awslogs-stream-prefix": "firelens"
    }
   },
   "memoryReservation": 50
   },
   {
    "essential": true,
    "image": "nginx",
    "name": "app",
    "logConfiguration": {
     "logDriver":"awsfirelens",
     "options": {
     "Name": "cloudwatch_logs",
     "region": "us-west-2",
     "log_key": "log",
```

```
                                "log_group_name": "/aws/ecs/containerinsights/my-
cluster/application",
     "auto_create_group": "true",
     "log_stream_name": "my-task-id"
    }
   },
   "memoryReservation": 100
  }
 ]
}
```

Setting up Container Insights on Red Hat OpenShift on AWS (ROSA)

CloudWatch Container Insights with enhanced observability supports Red Hat OpenShift on AWS (ROSA). This version supports enhanced observability for ROSA clusters. After you install the CloudWatch agent operator Helm chart, Container Insights auto-collects detailed infrastructure telemetry from the cluster level down to the container level in your environment. It then displays this performance data in curated dashboards removing the heavy lifting in observability setup.

> **Note**
>
> For Red Hat OpenShift on AWS (ROSA), when you install the CloudWatch agent operator using helm charts, the CloudWatch agent is by default also enabled to receive both metrics and traces from your applications that are instrumented for Application Signals. If you would like to optionally pass in custom configuration rules, you can do so by passing in a custom agent configuration by using the Helm chart, as outlined in (Optional) [Additional configuration], as outlined in (Optional) Additional configuration.

To install Container Insights with enhanced observability on a Red Hat OpenShift on AWS (ROSA) cluster

1. If necessary, install Helm. For more information, see Quickstart Guide in the Helm documentation.
2. Install the CloudWatch agent operator by entering the following commands. Replace *my-cluster-name* with the name of your cluster, and replace *my-cluster-region* with the Region that the cluster runs in.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
```

```
helm repo update aws-observability
helm install --wait --create-namespace \
  --namespace amazon-cloudwatch amazon-cloudwatch-observability \
  aws-observability/amazon-cloudwatch-observability \
  --set clusterName=my-cluster-name \
  --set region=my-cluster-region \
  --set k8sMode=ROSA
```

3. Set up authorization for the agent operator by following the steps in Option 1, Option 2, or Option 3 in Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

Viewing Container Insights metrics

After you have Container Insights set up and it is collecting metrics, you can view those metrics in the CloudWatch console.

For Container Insights metrics to appear on your dashboard, you must complete the Container Insights setup. For more information, see Container Insights.

This procedure explains how to view the metrics that Container Insights automatically generates from the collected log data. The rest of this section explains how to further dive into your data and use CloudWatch Logs Insights to see more metrics at more levels of granularity.

To view Container Insights metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**, **Container Insights**.
3. Use the drop-down boxes near the top to select the type of resource to view, as well as the specific resource.

You can set a CloudWatch alarm on any metric that Container Insights collects. For more information, see Using Amazon CloudWatch alarms

> **Note**
>
> If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard. If you have not already enabled Application Insights, you can do so by choosing

Auto-configure Application Insights below the performance view in the Container Insights dashboard.
For more information about Application Insights and containerized applications, see Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring.

Viewing the top contributors

For some of the views in Container Insights performance monitoring, you can also see the top contributors by memory or CPU, or the most recently active resources. This is available when you select any of the following dashboards in the drop-down box near the top of the page:

- ECS Services
- ECS Tasks
- EKS Namespaces
- EKS Services
- EKS Pods

When you are viewing one of these types of resources, the bottom of the page displays a table sorted initially by CPU usage. You can change it to sort by memory usage or recent activity. To see more about one of the rows in the table, you can select the checkbox next to that row and then choose **Actions** and choose one of the options in the **Actions** menu.

Using CloudWatch Logs Insights to view Container Insights data

Container Insights collects metrics by using performance log events with using embedded metric format. The logs are stored in CloudWatch Logs. CloudWatch generates several metrics automatically from the logs which you can view in the CloudWatch console. You can also do a deeper analysis of the performance data that is collected by using CloudWatch Logs Insights queries.

For more information about CloudWatch Logs Insights, see Analyze Log Data with CloudWatch Logs Insights. For more information about the log fields you can use in queries, see Container Insights performance log events for Amazon EKS and Kubernetes.

To use CloudWatch Logs Insights to query your container metric data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Logs**, **Logs Insights**.

   Near the top of the screen is the query editor. When you first open CloudWatch Logs Insights, this box contains a default query that returns the 20 most recent log events.

3. In the box above the query editor, select one of the Container Insights log groups to query. For the following example queries to work, the log group name must end with **performance**.

   When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in **Discovered fields** in the right pane. It also displays a bar graph of log events in this log group over time. This bar graph shows the distribution of events in the log group that matches your query and time range, not only the events displayed in the table.

4. In the query editor, replace the default query with the following query and choose **Run query**.

   ```
   STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
   | SORT avg_node_cpu_utilization DESC
   ```

   This query shows a list of nodes, sorted by average node CPU utilization.

5. To try another example, replace that query with another query and choose **Run query**. More sample queries are listed later on this page.

   ```
   STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by
    PodName
   | SORT avg_number_of_container_restarts DESC
   ```

   This query displays a list of your pods, sorted by average number of container restarts.

6. If you want to try another query, you can use include fields in the list at the right of the screen. For more information about query syntax, see CloudWatch Logs Insights Query Syntax.

To see lists of your resources

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Resources**.
3. The default view is a list of your resources being monitored by Container Insights, and alarms that you have set on these resources. To see a visual map of the resources, choose **Map view**.

4. From the map view, you can pause your pointer over any resource in the map to see basic metrics about that resource. You can choose any resource to see more detailed graphs about the resource.

Use case: Seeing task-level metrics in Amazon ECS containers

The following example illustrates how to use CloudWatch Logs Insights to dive deeper into your Container Insights logs. For more examples, see the blog Introducing Amazon CloudWatch Container Insights for Amazon ECS.

Container Insights does not automatically generate metrics at the Task level of granularity. The following query displays task-level metrics for CPU and memory usage.

```
stats avg(CpuUtilized) as CPU, avg(MemoryUtilized) as Mem by TaskId, ContainerName
| sort Mem, CPU desc
```

Other sample queries for Container Insights

List of your pods, sorted by average number of container restarts

```
STATS avg(number_of_container_restarts) as avg_number_of_container_restarts by PodName
| SORT avg_number_of_container_restarts DESC
```

Pods requested versus pods running

```
fields @timestamp, @message
| sort @timestamp desc
| filter Type="Pod"
| stats min(pod_number_of_containers) as requested,
 min(pod_number_of_running_containers) as running, ceil(avg(pod_number_of_containers-
pod_number_of_running_containers)) as pods_missing by kubernetes.pod_name
| sort pods_missing desc
```

Count of cluster node failures

```
stats avg(cluster_failed_node_count) as CountOfNodeFailures
| filter Type="Cluster"
| sort @timestamp desc
```

Application log errors by container name

```
stats count() as countoferrors by kubernetes.container_name
| filter stream="stderr"
| sort countoferrors desc
```

Metrics collected by Container Insights

Container Insights collects one set of metrics for Amazon ECS and AWS Fargate on Amazon ECS, and a different set for Amazon EKS, AWS Fargate on Amazon EKS, RedHat OpenShift on AWS (ROSA), and Kubernetes.

Metrics are not visible until the container tasks have been running for some time.

Topics

- Amazon ECS Container Insights with enhanced observability metrics
- Amazon ECS Container Insights metrics
- Amazon EKS and Kubernetes Container Insights with enhanced observability metrics
- Amazon EKS and Kubernetes Container Insights metrics
- Container Insights performance log reference
- Container Insights Prometheus metrics monitoring
- Integration with Application Insights
- Viewing Amazon ECS lifecycle events within Container Insights
- Troubleshooting Container Insights
- Building your own CloudWatch agent Docker image
- Deploying other CloudWatch agent features in your containers

Amazon ECS Container Insights with enhanced observability metrics

Container Insights with enhanced observability provides deeper visibility into containerized workloads by offering:

- Higher metrics granularity at both task and container levels
- Improved monitoring and troubleshooting capabilities
- Integration with CloudWatch Logs for:
  - Correlating metrics anomalies with log entries

- Performing faster root cause analysis
- Reducing resolution time for complex container issues

Use cases

Container Insights with enhanced observability extends the capabilities of standard Container Insights. It enables the following use cases:

- **Task-level troubleshooting** – Identify performance bottlenecks at the task level. Analyze task-level metrics and compare them with reserved resources to determine if tasks have sufficient processing capacity
- **Container-level resource optimization** – Track utilization against reservation levels to identify containers that are either resource-constrained or over-provisioned
- **Container health assessment** – Monitor restart counts and state transitions to detect unstable containers requiring intervention
- **Application performance monitoring** – Track how applications communicate with each other, monitor resource usage patterns, and optimize data storage performance
- **Operational monitoring** – Monitor deployments, track task sets for blue or green deployments, and maintain platform health through service metrics

For more information on Amazon ECS metrics, see Amazon ECS service utilization metrics use cases and for information on container insights with enhanced observability Amazon ECS Container Insights with enhanced observability metrics.

Container Insights also shows cluster, service, and daemon-wide statistics by averaging data across all tasks. This provides a higher-level view of your service and daemon health, assisting in both environment monitoring and capacity planning.

> **Note**
>
> Amazon ECS Managed Daemon metrics use the same `ECS/ContainerInsights` namespace and the same `ServiceName` dimension as service metrics. For daemon metrics, the `ServiceName` dimension value uses the format `daemon:`*`daemon-name`*. For example, a daemon named `my-daemon` has a `ServiceName` dimension value of `daemon:my-daemon`. All metrics in the table below that include the `ServiceName` dimension also apply to Managed Daemons.

The following table lists the metrics and dimensions that Container Insights with enhanced observability collects for Amazon ECS. These metrics are in the `ECS/ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights with enhanced observability. Metrics do not appear before Container Insights with enhanced observability has been set up completely. For more information, see Set up Container Insights with enhanced observability.

The following metrics are available for all launch types.

Metric name	Dimensions	Description
`ContainerInstanceCount`	`ClusterName`	The number of EC2 instances running the Amazon ECS agent that are registered with a cluster. This metric is collected only for container instances that are running Amazon ECS tasks in the cluster. It is not collected for empty container instances that do not have any Amazon ECS tasks. Unit: Count

`ContainerCpuUtilized`	`ClusterName`
	
	`ContainerName` , `TaskId`, `ServiceName` , `ClusterName`
	
	`ContainerName` , `TaskDefinitionFami`
		The CPU units used by containers in the resource that is specified by the dimension set that you're using.
		
		Also applies to Managed Daemons.

Metric name	Dimensions	Description
	ly ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	Unit: None
ContainerCpuReserved	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The CPU units reserved by containers in the resource that is specified by the dimension set that you're using. This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used. Also applies to Managed Daemons. Unit: None

Metric name	Dimensions	Description
ContainerCpuUtilization	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The total percentage of CPU units being used by containers in the resource that is specified by the dimension set that you're using. Unit: Percent
ContainerMemoryUtilized	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The memory being used by containers in the resource that is specified by the dimension set that you're using. Also applies to Managed Daemons. Unit: Megabytes

Metric name	Dimensions	Description
ContainerMemoryReserved	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The memory that is reserved by container s in the resource that is specified by the dimension set that you're using. This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used. Also applies to Managed Daemons. Unit: Megabytes

Metric name	Dimensions	Description
ContainerMemoryUtilization	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The total percentage of memory being used by containers in the resource that is specified by the dimension set that you're using. Also applies to Managed Daemons. Unit: Percent
ContainerNetworkRxBytes	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The number of bytes received by the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second

Metric name	Dimensions	Description
ContainerNetworkTxBytes	ClusterName ContainerName , TaskId, ServiceName , ClusterName ContainerName , TaskDefinitionFamily ,ClusterName , TaskId TaskDefinitionFamily ,ClusterName , ContainerName ServiceName , ClusterName , ContainerName	The number of bytes transmitted by the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second

Metric name	Dimensions	Description
ContainerStorageReadBytes	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName	The number of bytes read from storage on the container in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes
ContainerStorageWriteBytes	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName	The number of bytes written to storage in the container that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes

Metric name	Dimensions	Description
CpuUtilized	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The CPU units used by tasks in the resource that is specified by the dimension set that you're using. Also applies to Managed Daemons. Unit: None
CpuReserved	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The CPU units reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used. Also applies to Managed Daemons. Unit: None

Metric name	Dimensions	Description
DeploymentCount	ServiceName , ClusterName	The number of deployments in an Amazon ECS service. Unit: Count
DesiredTaskCount	ServiceName , ClusterName	The desired number of tasks for an Amazon ECS service. Unit: Count
EBSFilesystemSize	ClusterName ,TaskDefinitionFamily , VolumeName TaskDefinitionFamily ,ClusterName ServiceName , ClusterName	The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is allocated to the resources specified by the dimensions you're using. This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
EBSFilesystemUtilized	ClusterName ,TaskDefinitionFamily , VolumeName TaskDefinitionFamily ,ClusterName ServiceName , ClusterName	The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is being used by the resources specified by the dimensions that you're using. This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later. For tasks run on Fargate, Fargate reserves space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this additional storage using tools like df. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
TaskEBSFilesystemUtilization	TaskDefinitionFamily ,ClusterName ClusterName , ServiceName ClusterName ClusterName , ServiceName ,TaskId TaskDefinitionFamily ,ClusterName , TaskId	The percentage of Amazon EBS filesystem storage that is being used by the task specified by the dimensions that you're using. This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later. Also applies to Managed Daemons. Unit: Percent

Metric name	Dimensions	Description
EphemeralStorageReserved [1]	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes reserved from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task. This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
EphemeralStorageUtilized [1]	ClusterName ClusterName , TaskDefinitionFamily ClusterName , ServiceName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes used from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task. This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
MemoryUtilized	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The memory being used by tasks in the resource that is specified by the dimension set that you're using. Also applies to Managed Daemons. Unit: Megabytes
MemoryReserved	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used. Also applies to Managed Daemons. Unit: Megabytes

Metric name	Dimensions	Description
NetworkRxBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes received by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second

Metric name	Dimensions	Description
NetworkTxBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second
PendingTaskCount	ServiceName , ClusterName	The number of tasks currently in the PENDING state. Unit: Count
RunningTaskCount	ServiceName , ClusterName	The number of tasks currently in the RUNNING state. Unit: Count

Metric name	Dimensions	Description
RestartCount	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId ClusterName , ServiceName , ContainerName ClusterName , ServiceName , TaskId, ContainerName TaskDefinitionFamily , ClusterName , ContainerName TaskDefinitionFamily , ClusterName , TaskId, ContainerName	The number of times a container in an Amazon ECS task has been restarted. This metric is collected only for containers that have a restart policy enabled. Also applies to Managed Daemons. Unit: Count

Metric name	Dimensions	Description
UnHealthyContainerHealthStatus	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName	The number of unhealthy containers based on container health check status. A container is considered unhealthy when its health check returns an unhealthy status. This metric is collected only for containers that have a health check configured in the task definition. The metric value is 1 when the container health status is UNHEALTHY , and 0 when the health status is HEALTHY. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count

Metric name	Dimensions	Description
StorageReadBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes read from storage on the instance in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes
StorageWriteBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The number of bytes written to storage in the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes
TaskCount	ClusterName	The number of tasks running in the cluster. Unit: Count

Metric name	Dimensions	Description
TaskCpuUtilization	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The total percentage of CPU units being used by a task. Also applies to Managed Daemons. Unit: Percent
TaskEphemeralStorageUtilization	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The total percentage of ephemeral storage being used by a task. Also applies to Managed Daemons. Unit: Percent

Metric name	Dimensions	Description
TaskMemoryUtilization	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId	The total percentage of memory being used by a task. Also applies to Managed Daemons. Unit: Percent
TaskSetCount	ServiceName , ClusterName	The number of task sets in the service. Unit: Count

Metric name	Dimensions	Description
ContainerGPUUtilization	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The percentage utilization of the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Percent

Metric name	Dimensions	Description
ContainerGPUMemoryUtilization	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The percentage of frame buffer memory used on the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Percent

Metric name	Dimensions	Description
ContainerGPUMemoryTotal	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Bytes

Metric name	Dimensions	Description
ContainerGPUMemoryUsed	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The frame buffer memory, in bytes, used on the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Bytes

Metric name	Dimensions	Description
ContainerGPUPowerDraw	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The power usage, in watts, of the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Watts

Metric name	Dimensions	Description
ContainerGPUTemperature	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The temperature, in degrees Celsius, of the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Celsius

Metric name	Dimensions	Description
ContainerGPURestartAppXidCount	ClusterName ClusterName , ServiceName , ContainerName ClusterName , TaskDefinitionFamily , ContainerName ClusterName , ServiceName , TaskId, ContainerName ClusterName , TaskDefinitionFamily , TaskId, ContainerName ClusterName , ServiceName , TaskId, ContainerName , AcceleratedDevice	The count of NVIDIA Xid errors whose immediate-action resolution bucket is RESTART_APP , observed on the GPU(s) allocated to the container. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Count

Metric name	Dimensions	Description
TaskGPUUtilization	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId ClusterName , ServiceName ,TaskId, AcceleratedDevice	The percentage utilization of the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Percent

Metric name	Dimensions	Description
TaskGPUMemoryUtilization	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId ClusterName , ServiceName , TaskId, AcceleratedDevice	The percentage of frame buffer memory used on the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Percent

Metric name	Dimensions	Description
TaskGPUMemoryTotal	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId ClusterName , ServiceName ,TaskId, AcceleratedDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Bytes

Metric name	Dimensions	Description
TaskGPUMemoryUsed	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName ,TaskId ClusterName , TaskDefinitionFamily ,TaskId ClusterName , ServiceName ,TaskId, AcceleratedDevice	The frame buffer memory, in bytes, used on the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Bytes

Metric name	Dimensions	Description
TaskGPUPowerDraw	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId ClusterName , ServiceName , TaskId, AcceleratedDevice	The power usage, in watts, of the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Watts

Metric name	Dimensions	Description
TaskGPUTemperature	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId ClusterName , ServiceName , TaskId, AcceleratedDevice	The temperature, in degrees Celsius, of the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Celsius

Metric name	Dimensions	Description
TaskGPURestartAppXidCount	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily ClusterName , ServiceName , TaskId ClusterName , TaskDefinitionFamily , TaskId ClusterName , ServiceName , TaskId, AcceleratedDevice	The count of NVIDIA Xid errors whose immediate-action resolution bucket is RESTART_APP , observed on the GPU(s) allocated to the task. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Count
InstanceGPULimit	ClusterName ClusterName , CapacityProviderName ClusterName , CapacityProviderName , ContainerInstanceId , EC2InstanceId	The total number of GPUs available on the instance. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Count

Metric name	Dimensions	Description
InstanceGPUUsageTotal	ClusterName ClusterName , CapacityProviderName ClusterName , CapacityProviderName , ContainerInstanceId , EC2InstanceId	The number of GPUs currently allocated to running tasks on the instance. Available only for Amazon ECS Managed Instances running NVIDIA GPU-enabled Amazon EC2 instance types. Unit: Count

Note

The EphemeralStorageReserved and EphemeralStorageUtilized metrics are only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as df.

The following metrics are available when you complete the steps in Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS and use the EC2 launch type.

Metric name	Dimensions	Description
instance_cpu_limit	ClusterName	The maximum number of CPU units that can be assigned to a single EC2 instance in the cluster. Unit: None

Metric name	Dimensions	Description
instance_cpu_reserved_capacity	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The percentage of CPU currently being reserved on a single EC2 instance in the cluster. Unit: Percent
instance_cpu_usage_total	ClusterName	The number of CPU units being used on a Single EC2 instance in the cluster. Unit: None
instance_cpu_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of CPU units being used on a single EC2 instance in the cluster. Unit: Percent
instance_filesystem_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of file system capacity being used on a single EC2 instance in the cluster. Unit: Percent

Metric name	Dimensions	Description
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 instance in this cluster. Unit: Bytes
instance_memory_reserved_capacity	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The percentage of Memory currently being reserved on a single EC2 instance in the cluster. Unit: Percent
instance_memory_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of memory being used on a single EC2 instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Percent

Metric name	Dimensions	Description
instance_memory_working_set	ClusterName	The amount of memory, in bytes, being used on a single EC2 instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Bytes
instance_network_total_bytes	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 instance in the cluster. Unit: Bytes/second
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 instance in the cluster. Unit: Count

Amazon ECS Container Insights metrics

Container Insights metrics provides additional network, storage, and ephemeral storage metrics. These metrics provide more information than standard Amazon ECS metrics. Container Insights integrates with CloudWatch Logs. You can correlate metric changes with log entries for easier troubleshooting. Container Insights also shows cluster, service, and daemon-wide statistics by averaging data across all tasks. This provides a higher-level view of your service and daemon health, assisting in both environment monitoring and capacity planning.

Use cases

- **Problem identification and troubleshooting** – Track failed deployments by analyzing task state transition patterns, enabling rapid identification of failure points. Diagnose configuration issues through comprehensive examination of task startup sequences and initialization behaviors
- **Cluster and Service-Level health assessment** – Shows average task performance across the cluster. This approach moderates outliers to deliver a more stable view of cluster and service health. Use these insights for general service monitoring where extreme values could be misleading
- **Service availability issues** – Detect deployment failures by monitoring running task count metrics. Correlate service event logs with performance metrics to understand infrastructure impacts. Track task restart patterns to identify unstable services or infrastructure issues
- **Capacity planning for average load** – It helps determine resource requirements based on typical task behavior patterns, provides consistent metrics that support effective long-term planning, and reduces the impact of short-lived spikes on capacity decisions
- **Provides additional metrics** – Collects additional network, storage, and ephemeral storage metrics not available in vended metrics

For more information on Amazon ECS metrics, see Amazon ECS service utilization metrics use cases and for information on container insights with enhanced observability Amazon ECS Container Insights with enhanced observability metrics.

> **Note**
>
> Amazon ECS Managed Daemon metrics use the same `ECS/ContainerInsights` namespace and the same `ServiceName` dimension as service metrics. For daemon metrics, the `ServiceName` dimension value uses the format `daemon:`*`daemon-name`*. For example, a daemon named `my-daemon` has a `ServiceName` dimension value of `daemon:my-`

daemon. All metrics in the table below that include the `ServiceName` dimension also apply to Managed Daemons.

The following table lists the metrics and dimensions that Container Insights collects for Amazon ECS. These metrics are in the `ECS/ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see Container Insights.

The following metrics are available when you complete the steps in Setting up Container Insights on Amazon ECS.

Metric name	Dimensions	Description
`ContainerInstanceCount`	`ClusterName`	The number of EC2 instances running the Amazon ECS agent that are registered with a cluster. This metric is collected only for container instances that are running Amazon ECS tasks in the cluster. It is not collected for empty container instances that do not have any Amazon ECS tasks. Unit: Count
`CpuUtilized`	`TaskDefinitionFamily` , `ClusterName`	The CPU units used by tasks in the resource that is specified by the

Metric name	Dimensions	Description

	`ServiceName` , `ClusterName`
	
	`ClusterName`
		dimension set that you're using.
		
		Also applies to Managed Daemons.
		
		Unit: None

`CpuReserved`	`TaskDefinitionFamily` , `ClusterName`
	
	`ServiceName` , `ClusterName`
	
	`ClusterName`
		The CPU units reserved by tasks in the resource that is specified by the dimension set that you're using.
		
		This metric is collected based on the CPU reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance CPU reservation is used.
		
		Also applies to Managed Daemons.
		
		Unit: None

`DeploymentCount`	`ServiceName` , `ClusterName`	The number of deployments in an Amazon ECS service. Unit: Count

Metric name	Dimensions	Description
DesiredTaskCount	ServiceName , ClusterName	The desired number of tasks for an Amazon ECS service. Unit: Count
EBSFilesystemSize	VolumeName , TaskDefinitionFamily ,ClusterName TaskDefinitionFamily ,ClusterName ServiceName , ClusterName	The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is allocated to the resources specified by the dimensions you're using. This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
EBSFilesystemUtilized	VolumeName , TaskDefinitionFamily ,ClusterName TaskDefinitionFamily ,ClusterName ServiceName , ClusterName	The total amount, in gigabytes (GB), of Amazon EBS filesystem storage that is being used by the resources specified by the dimensions that you're using. This metric is only available for tasks that run on Amazon ECS infrastructure running on Fargate using platform version 1.4.0 or Amazon EC2 instances using container agent version 1.79.0 or later. For tasks run on Fargate, Fargate reserves space on the disk that is only used by Fargate. There is no cost associated with the space Fargate uses, but you will see this additional storage using tools like df. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
EphemeralStorageReserved [1]	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The number of bytes reserved from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task. This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)

Metric name	Dimensions	Description
EphemeralStorageUtilized [1]	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The number of bytes used from ephemeral storage in the resource that is specified by the dimensions that you're using. Ephemeral storage is used for the container root filesystem and any bind mount host volumes defined in the container image and task definition. The amount of ephemeral storage can't be changed in a running task. This metric is only available for tasks that run on Fargate Linux platform version 1.4.0 or later. Also applies to Managed Daemons. Unit: Gigabytes (GB)
InstanceOSFilesystemUtilization	CapacityProviderName , ClusterName , ContainerInstanceId , EC2InstanceId ClusterName	The percentage of total disk space that is used for OS volume.

Metric name	Dimensions	Description
InstanceDataFilesystemUtilization	CapacityProviderName ,ClusterName , ContainerInstanceId ,EC2InstanceId ClusterName	The percentage of total disk space that is used for data volume.

Metric name	Dimensions	Description
MemoryUtilized	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The memory being used by tasks in the resource that is specified by the dimension set that you're using. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Although MemoryUtilized and MemoryReserved are marked as "Megabytes", the actual units are in MiB (Mebibytes). Also applies to Managed Daemons. Unit: Megabytes

Metric name	Dimensions	Description
MemoryReserved	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The memory that is reserved by tasks in the resource that is specified by the dimension set that you're using. This metric is collected based on the memory reservation defined in the task definition, for example, at the task or all containers level. If this is not specified in the task definition, then the instance memory reservation is used. Also applies to Managed Daemons. Unit: Megabytes **Note** Although MemoryUtilized and MemoryReserved are marked as "Megabytes", the actual units are in MiB (Mebibytes).

Metric name	Dimensions	Description
NetworkRxBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The number of bytes received by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second

Metric name	Dimensions	Description
NetworkTxBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The number of bytes transmitted by the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. This metric is available only for containers in tasks using the awsvpc or bridge network modes. Also applies to Managed Daemons. Unit: Bytes/Second
PendingTaskCount	ServiceName , ClusterName	The number of tasks currently in the PENDING state. Unit: Count
RunningTaskCount	ServiceName , ClusterName	The number of tasks currently in the RUNNING state. Unit: Count

Metric name	Dimensions	Description
RestartCount	ClusterName ClusterName , ServiceName ClusterName , TaskDefinitionFamily	The number of times a container in an Amazon ECS task has been restarted. This metric is collected only for containers that have a restart policy enabled. Also applies to Managed Daemons. Unit: Count
ServiceCount	ClusterName	The number of services in the cluster. Unit: Count
StorageReadBytes	TaskDefinitionFamily ,ClusterName ServiceName , ClusterName ClusterName	The number of bytes read from storage on the instance in the resource that is specified by the dimensions that you're using. This does not include read bytes for your storage devices. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes

Metric name	Dimensions	Description
StorageWriteBytes	TaskDefinitionFamily , ClusterName ServiceName , ClusterName ClusterName	The number of bytes written to storage in the resource that is specified by the dimensions that you're using. This metric is obtained from the Docker runtime. Also applies to Managed Daemons. Unit: Bytes
TaskCount	ClusterName	The number of tasks running in the cluster. Unit: Count
TaskSetCount	ServiceName , ClusterName	The number of task sets in the service. Unit: Count

Note

The EphemeralStorageReserved and EphemeralStorageUtilized metrics are only available for tasks that run on Fargate Linux platform version 1.4.0 or later.
Fargate reserves space on disk. It is only used by Fargate. You aren't billed for it. It isn't shown in these metrics. However, you can see this additional storage in other tools such as df.

The following metrics are available when you complete the steps in Deploying the CloudWatch agent to collect EC2 instance-level metrics on Amazon ECS

Metric name	Dimensions	Description
instance_cpu_limit	ClusterName	The maximum number of CPU units that can be assigned to a single EC2 Instance in the cluster. Unit: None
instance_cpu_reserved_capacity	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The percentage of CPU currently being reserved on a single EC2 instance in the cluster. Unit: Percent
instance_cpu_usage_total	ClusterName	The number of CPU units being used on a Single EC2 instance in the cluster. Unit: None
instance_cpu_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of CPU units being used on a single EC2 instance in the cluster. Unit: Percent
instance_filesystem_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of file system capacity being used on a single

Metric name	Dimensions	Description
		EC2 instance in the cluster. Unit: Percent
instance_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single EC2 Instance in this cluster. Unit: Bytes
instance_memory_reserved_capacity	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The percentage of Memory currently being reserved on a single EC2 Instance in the cluster. Unit: Percent

Metric name	Dimensions	Description
instance_memory_utilization	ClusterName InstanceId ,ContainerInstanceId , ClusterName	The total percentage of memory being used on a single EC2 Instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Percent

Metric name	Dimensions	Description
instance_memory_working_set	ClusterName	The amount of memory, in bytes, being used on a single EC2 Instance in the cluster. **Note** If you're using the Java ZGC garbage collector for your application, this metric might be inaccurate. Unit: Bytes
instance_network_total_bytes	ClusterName	The total number of bytes per second transmitted and received over the network on a single EC2 Instance in the cluster. Unit: Bytes/second
instance_number_of_running_tasks	ClusterName	The number of running tasks on a single EC2 Instance in the cluster. Unit: Count

Amazon EKS and Kubernetes Container Insights with enhanced observability metrics

The following tables list the metrics and dimensions that Container Insights with enhanced observability collects for Amazon EKS and Kubernetes. These metrics are in the `ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights with enhanced observability metrics in your console, be sure that you have completed the setup of Container Insights with enhanced observability. Metrics do not appear before Container Insights with enhanced observability has been set up completely. For more information, see Container Insights.

If you are using version 1.5.0 or later of the Amazon EKS add-on or version 1.300035.0 of the CloudWatch agent, most metrics listed in the following table are collected for both Linux and Windows nodes. See the **Metric Name** column of the table to see which metrics are not collected for Windows.

With the earlier version of Container Insights which delivers aggregated metrics at Cluster and Service level, the metrics are charged as custom metrics. With Container Insights with enhanced observability for Amazon EKS, Container Insights metrics are charged per observation instead of being charged per metric stored or log ingested. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

> **Note**
>
> On Windows, network metrics such as `pod_network_rx_bytes` and `pod_network_tx_bytes` are not collected for host process containers.
> On RedHat OpenShift on AWS (ROSA) clusters , diskio metrics such as `node_diskio_io_serviced_total` and `node_diskio_io_service_bytes_total` are not collected.

Metric name	Dimensions	Description
`cluster_failed_node_count`	`ClusterName`	The number of failed worker nodes in the cluster. A node is considered failed if it is suffering from any *node*

Metric name	Dimensions	Description
		conditions. For more informati on, see Conditions in the Kubernetes documentation.
cluster_node_count	ClusterName	The total number of worker nodes in the cluster.
namespace_number_o f_running_pods	Namespace ClusterName ClusterName	The number of pods running per namespace in the resource that is specified by the dimensions that you're using.
node_cpu_limit	ClusterName ClusterName ,InstanceI d ,NodeName	The maximum number of CPU units that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description
node_cpu_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker. Formula: node_cpu_request / node_cpu_limit **Note** node_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_cpu_usage_total	ClusterName ClusterName , InstanceId , NodeName	The number of CPU units being used on the nodes in the cluster.
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName	The total percentage of CPU units being used on the nodes in the cluster. Formula: node_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description
node_filesystem_utilization	NodeName, ClusterName , InstanceId ClusterName	The total percentage of file system capacity being used on nodes in the cluster. Formula: node_filesystem_usage / node_filesystem_capacity **Note** node_filesystem_usage and node_filesystem_capacity are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_limit	ClusterName ClusterName , InstanceId , NodeName	The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description
node_filesystem_inodes It is not available on Windows.	ClusterName ClusterName ,InstanceId ,NodeName	The total number of inodes (used and unused) on a node.
node_filesystem_inodes_free It is not available on Windows.	ClusterName ClusterName ,InstanceId ,NodeName	The number of unused inodes on a node.
node_gpu_limit	ClusterName ClusterName ,InstanceId ,NodeName	The total number of GPU(s) available on the node.
node_gpu_usage_total	ClusterName ClusterName ,InstanceId ,NodeName	The number of GPU(s) being used by the running pods on the node.
node_gpu_reserved_capacity	ClusterName ClusterName ,InstanceId ,NodeName	

Metric name	Dimensions	Description
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being reserved on the nodes in the cluster. Formula: node_memory_request / node_memory_limit **Note** node_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used by the node or nodes. It is the percentage of node memory usage divided by the node memory limitation. Formula: node_memory_working_set / node_memory_limit .

Metric name	Dimensions	Description
node_memory_working_set	ClusterName ClusterName , InstanceId , NodeName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName	The total number of bytes per second transmitted and received over the network per node in a cluster. Formula: node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes and node_network_tx_bytes are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName	The number of running containers per node in a cluster.

Metric name	Dimensions	Description
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName	The number of running pods per node in a cluster.
node_status_allocatable_pods	ClusterName ClusterName ,InstanceId ,NodeName	The number of pods that can be assigned to a node based on its allocatable resources, which is defined as the remainder of a node's capacity after accounting for system daemons reservations and hard eviction thresholds.
node_status_capacity_pods	ClusterName ClusterName ,InstanceId ,NodeName	The number of pods that can be assigned to a node based on its capacity.
node_status_condition_ready	ClusterName ClusterName ,InstanceId ,NodeName	Indicates whether the node status condition Ready is true for Amazon EC2 nodes.
node_status_condition_memory_pressure	ClusterName ClusterName ,InstanceId ,NodeName	Indicates whether the node status condition MemoryPressure is true.
node_status_condition_pid_pressure	ClusterName ClusterName ,InstanceId ,NodeName	Indicates whether the node status condition PIDPressure is true.
node_status_condition_disk_pressure	ClusterName ClusterName ,InstanceId ,NodeName	Indicates whether the node status condition OutOfDisk is true.

Metric name	Dimensions	Description
node_status_condition_unknown	ClusterName ClusterName , InstanceId , NodeName	Indicates whether any of the node status conditions are Unknown.
node_interface_network_rx_dropped	ClusterName ClusterName , InstanceId , NodeName	The number of packets which were received and subsequently dropped by a network interface on the node.
node_interface_network_tx_dropped	ClusterName ClusterName , InstanceId , NodeName	The number of packets which were due to be transmitted but were dropped by a network interface on the node.
node_diskio_io_service_bytes_total It is not available on Windows or on ROSA clusters.	ClusterName ClusterName , InstanceId , NodeName	The total number of bytes transferred by all I/O operations on the node.
node_diskio_io_serviced_total It is not available on Windows or on ROSA clusters.	ClusterName ClusterName , InstanceId , NodeName	The total number of I/O operations on the node.

Metric name	Dimensions	Description
pod_cpu_reserved_capacity	PodName, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	The CPU capacity that is reserved per pod in a cluster. Formula: pod_cpu_request / node_cpu_limit **Note** pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_cpu_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of CPU units being used by pods. Formula: pod_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description
pod_cpu_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of CPU units being used by pods relative to the pod limit. Formula: pod_cpu_usage_total / pod_cpu_limit
pod_memory_reserved_capacity	PodName, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , Service	The percentage of memory that is reserved for pods. Formula: pod_memory_request / node_memory_limit **Note** pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of memory currently being used by the pod or pods. Formula: pod_memory_working_set / node_memory_limit
pod_memory_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The percentage of memory that is being used by pods relative to the pod limit. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: pod_memory_working_set / pod_memory_limit

Metric name	Dimensions	Description
pod_network_rx_bytes	PodName, Namespace , ClusterName Namespace ,ClusterName Service, Namespace , ClusterName ClusterName ClusterName ,Namespace , PodName, FullPodName	The number of bytes per second being received over the network by the pod. Formula: sum(pod_interface_network_rx_bytes) **Note** pod_interface_network_rx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_network_tx_bytes	PodName, Namespace , ClusterName Namespace, ClusterName Service, Namespace , ClusterName ClusterName ClusterName , Namespace , PodName, FullPodName	The number of bytes per second being transmitted over the network by the pod. Formula: sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_cpu_request	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	The CPU requests for the pod. Formula: sum(container_cpu_request) **Note** pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_request	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	The memory requests for the pod. Formula: sum(container_memory_request) **Note** pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_cpu_limit	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	The CPU limit defined for the containers in the pod. If any containers in the pod don't have a CPU limit defined, this metric doesn't appear. Formula: sum(container_cpu_limit) **Note** pod_cpu_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_limit	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	The memory limit defined for the containers in the pod. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: sum(container_memory_limit) **Note** pod_cpu_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_status_failed	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Indicates that all containers in the pod have terminated, and at least one container has terminated with a non-zero status or was terminated by the system.

Metric name	Dimensions	Description
pod_status_ready	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Indicates that all containers in the pod are ready, having reached the condition of ContainerReady .
pod_status_running	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Indicates that all containers in the pod are running.
pod_status_scheduled	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Indicates that the pod has been scheduled to a node.

Metric name	Dimensions	Description
pod_status_unknown	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Indicates that status of the pod can't be obtained.
pod_status_pending	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Indicates that the pod has been accepted by the cluster but one or more of the containers has not become ready yet.
pod_status_succeeded	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Indicates that all containers in the pod have successfully terminated and will not be restarted.

Metric name	Dimensions	Description
pod_number_of_containers	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers defined in the pod specification.
pod_number_of_running_containers	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers in the pod which are currently in the Running state.
pod_container_status_terminated	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Terminated state.

Metric name	Dimensions	Description
pod_container_status_running	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Running state.
pod_container_status_waiting	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers in the pod which are in the Waiting state.
pod_container_status_waiting_reason_crash_loop_back_off	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending because of a CrashLoopBackOff error, where a container repeatedly fails to start.

Metric name	Dimensions	Description
pod_container_status_waiting_reason_create_container_config_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason CreateContainerConfigError . This is because of an error while creating the container configuration.
pod_container_status_waiting_reason_create_container_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason CreateContainerError because of an error while creating the container.
pod_container_status_waiting_reason_image_pull_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending because of ErrImagePull , ImagePullBackOff , or InvalidImageName . These situations are because of an error while pulling the container image.

Metric name	Dimensions	Description
pod_container_status_waiting_reason_start_error	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	Reports the number of containers in the pod which are pending with the reason being StartError because of an error while starting the container.
pod_container_status_terminated_reason_oom_killed	ContainerName , FullPodName , PodName, Namespace , ClusterName ContainerName , PodName, Namespace , ClusterName ClusterName	Indicates a pod was terminated for exceeding the memory limit. This metric is only displayed when this issue occurs.
pod_interface_network_rx_dropped	ClusterName PodName, Namespace , ClusterName Namespace , ClusterName , Service ClusterName , Namespace , PodName, FullPodName	The number of packets which were received and subsequently dropped a network interface for the pod.

Metric name	Dimensions	Description
pod_interface_network_tx_dropped	ClusterName PodName, Namespace , ClusterName Namespace ,ClusterName ,Service ClusterName ,Namespace ,PodName, FullPodName	The number of packets which were due to be transmitted but were dropped for the pod.
pod_memory_working_set	ClusterName ClusterName ,Namespace ,PodName ClusterName ,Namespace ,Service ClusterName ,Namespace ,PodName, FullPodName	The memory in bytes that is currently being used by a pod.
pod_cpu_usage_total	ClusterName ClusterName ,Namespace ,PodName ClusterName ,Namespace ,Service ClusterName ,Namespace ,PodName, FullPodName	The number of CPU units used by a pod.

Metric name	Dimensions	Description
container_cpu_utilization	ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	The percentage of CPU units being used by the container. Formula: container_cpu_usage_total / node_cpu_limit **Note** container_cpu_utilization is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
container_cpu_utilization_over_container_limit	ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	The percentage of CPU units being used by the container relative to the container limit. If the container doesn't have a CPU limit defined, this metric doesn't appear. Formula: container_cpu_usage_total / container_cpu_limit **Note** container_cpu_utilization_over_container_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
container_memory_utilization	ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	The percentage of memory units being used by the container. Formula: container_memory_working_set / node_memory_limit **Note** container_memory_utilization is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
container_memory_utilization_over_container_limit	ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	The percentage of memory units being used by the container relative to the container limit. If the container doesn't have a memory limit defined, this metric doesn't appear. Formula: container_memory_working_set / container_memory_limit **Note** container_memory_utilization_over_container_limit is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
container_memory_failures_total It is not available on Windows.	ClusterName PodName, Namespace , ClusterName , ContainerName PodName, Namespace , ClusterName , ContainerName , FullPodName	The number of memory allocation failures experienced by the container.
pod_number_of_container_restarts	PodName, Namespace , ClusterName	The total number of container restarts in a pod.
service_number_of_running_pods	Service, Namespace , ClusterName ClusterName	The number of pods running the service or services in the cluster.
replicas_desired	ClusterName PodName, Namespace , ClusterName	The number of pods desired for a workload as defined in the workload specification.
replicas_ready	ClusterName PodName, Namespace , ClusterName	The number of pods for a workload that have reached the ready status.
status_replicas_available	ClusterName PodName, Namespace , ClusterName	The number of pods for a workload which are available . A pod is available when it has been ready for the minReadySeconds defined in the workload specification.

Metric name	Dimensions	Description
status_replicas_unavailable	ClusterName PodName, Namespace , ClusterName	The number of pods for a workload which are unavailable. A pod is available when it has been ready for the minReadySeconds defined in the workload specification. Pods are unavailable if they have not met this criterion.
apiserver_storage_objects	ClusterName ClusterName , resource	The number of objects stored in etcd at the time of the last check.
apiserver_storage_db_total_size_in_bytes	ClusterName ClusterName , endpoint	Total size of the storage database file physically allocated in bytes. This metric is experimental and might change in future releases of Kubernetes. Unit: Bytes Meaningful statistics: Sum, Average, Minimum, Maximum
apiserver_request_total	ClusterName ClusterName , code, verb	The total number of API requests to the Kubernetes API server.
apiserver_request_duration_seconds	ClusterName ClusterName , verb	Responce latency for API requests to the Kubernetes API server.

Metric name	Dimensions	Description
apiserver_admission_controller_admission_duration_seconds	ClusterName ClusterName ,operation	Admission controller latency in seconds. An admission controller is code which intercepts requests to the Kubernetes API server.
rest_client_request_duration_seconds	ClusterName ClusterName ,operation	Reponse latency experienced by clients calling the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.
rest_client_requests_total	ClusterName ClusterName ,code, method	The total number of API requests to the Kubernetes API server made by clients. This metric is experimental and may change in future releases of Kubernetes.
etcd_request_duration_seconds	ClusterName ClusterName ,operation	Response latency of API calls to Etcd. This metric is experimental and may change in future releases of Kubernetes.
apiserver_storage_size_bytes	ClusterName ClusterName ,endpoint	Size of the storage database file physically allocated in bytes. This metric is experimental and may change in future releases of Kubernetes.
apiserver_longrunning_requests	ClusterName ClusterName ,resource	The number of active long-running requests to the Kubernetes API server.

Metric name	Dimensions	Description
apiserver_current_inflight_requests	ClusterName ClusterName , request_kind	The number of requests that are being processed by Kubernetes API server.
apiserver_admission_webhook_admission_duration_seconds	ClusterName ClusterName , name	Admission webhook latency in seconds. Admission webhooks are HTTP callbacks that receive admission requests and do something with them.
apiserver_admission_step_admission_duration_seconds	ClusterName ClusterName , operation	Admission sub-step latency in seconds.
apiserver_requested_deprecated_apis	ClusterName ClusterName , group	Number of requests to deprecated APIs on the Kubernetes API server.
apiserver_request_total_5xx	ClusterName ClusterName , code, verb	Number of requests to the Kubernetes API server which were responded to with a 5XX HTTP response code.
apiserver_storage_list_duration_seconds	ClusterName ClusterName , resource	Response latency of listing objects from Etc. This metric is experimental and may change in future releases of Kubernetes.
apiserver_flowcontrol_request_concurrency_limit	ClusterName ClusterName , priority_level	The number of threads used by the currently executing requests in the API Priority and Fairness subsystem.

Metric name	Dimensions	Description
apiserver_flowcontrol_rejected_requests_total	ClusterName ClusterName ,reason	Number of requests rejected by API Priority and Fairness subsystem. This metric is experimental and may change in future releases of Kubernetes.
apiserver_current_inqueue_requests	ClusterName ClusterName ,request_kind	The number queued requests queued by the Kubernetes API server. This metric is experimental and may change in future releases of Kubernetes.

NVIDIA GPU metrics

Beginning with version `1.300034.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects NVIDIA GPU metrics from EKS workloads by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version `v1.3.0-eksbuild.1` or later. For more information, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. These NVIDIA GPU metrics that are collected are listed in the table in this section.

For Container Insights to collect NVIDIA GPU metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.3.0-eksbuild.1` or later.
- The NVIDIA device plugin for Kubernetes must be installed in the cluster.
- The NVIDIA container toolkit must be installed on the nodes of the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

You can opt out of collecting NVIDIA GPU metrics by setting the `accelerated_compute_metrics` option in the beginn CloudWatch agent configuration file to `false`. For more information and an example opt-out configuration, see (Optional) Additional configuration.

Metric name	Dimensions	Description
container_gpu_memory_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the container.
container_gpu_memory_used	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the container.
container_gpu_memory_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName	The percentage of frame buffer used of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	
container_gpu_power_draw	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The power usage in watts of the GPU(s) allocated to the container.
container_gpu_temperature	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the container.

Metric name	Dimensions	Description
container_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the GPU(s) allocated to the container.
container_gpu_tensor_core_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the tensor cores on the GPU(s) allocated to the container.
node_gpu_memory_total	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the node.

Metric name	Dimensions	Description
node_gpu_memory_used	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the node.
node_gpu_memory_utilization	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The percentage of frame buffer used on the GPU(s) allocated to the node.
node_gpu_power_draw	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The power usage in watts of the GPU(s) allocated to the node.
node_gpu_temperature	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The temperature in degrees celsius of the GPU(s) allocated to the node.

Metric name	Dimensions	Description
node_gpu_utilization	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The percentage utilization of the GPU(s) allocated to the node.
node_gpu_tensor_core_utilization	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, GpuDevice	The percentage utilization of the tensor cores on the GPU(s) allocated to the node.
pod_gpu_memory_total	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName . GpuDevice	The total frame buffer size, in bytes, on the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_memory_used	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The bytes of frame buffer used on the GPU(s) allocated to the pod.
pod_gpu_memory_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The percentage of frame buffer used of the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_power_draw	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The power usage in watts of the GPU(s) allocated to the pod.
pod_gpu_temperature	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName . GpuDevice	The temperature in degrees Celsius of the GPU(s) allocated to the pod.

Metric name	Dimensions	Description
pod_gpu_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the GPU(s) allocated to the pod.
pod_gpu_tensor_core_utilization	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , GpuDevice	The percentage utilization of the tensor cores on the GPU(s) allocated to the pod.

Detailed GPU monitoring

Beginning with version 1.300062.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS supports detailed GPU monitoring with sub-minute collection intervals. This addresses monitoring gaps for short-duration machine learning inference workloads that may be completely missed by standard collection intervals. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version v4.7.0-eksbuild.1 or later. For more information, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

By default, GPU metrics are collected and ingested at 60-second intervals. With detailed monitoring enabled, the CloudWatch agent collect GPU metrics at sub-minute intervals (minimum 1 second), but metrics are still ingested to CloudWatch at 1-minute intervals. However, you can query statistical aggregations (such as minimum, maximum, and percentiles like p90) of the sub-minute datapoints within each 1-minute period, providing accurate GPU utilization data and better resource optimization.

Configuration

To enable detailed GPU monitoring, update your CloudWatch agent configuration to include the `accelerated_compute_gpu_metrics_collection_interval` parameter in the `kubernetes` section, as in the following example.

```
{
    "logs": {
        "metrics_collected": {
            "kubernetes": {
                "cluster_name": "MyCluster",
                "enhanced_container_insights": true,
                "accelerated_compute_metrics": true,
                "accelerated_compute_gpu_metrics_collection_interval": 1
            }
        }
    }
}
```

The `accelerated_compute_gpu_metrics_collection_interval` parameter accepts values in seconds, with a minimum value of 1 second. Setting it to 1 enables 1-second collection intervals. If this parameter is not specified, the default 60-second interval is used.

For complete configuration instructions, see Setup guide (AWS CLI).

AWS Neuron metrics for AWS Trainium and AWS Inferentia

Beginning with version `1.300036.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects accelerated computing metrics from AWS Trainium and AWS Inferentia accelerators by default. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version `v1.5.0-eksbuild.1` or later. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about AWS Trainium, see AWS Trainium. For more information about AWS Inferentia, see AWS Inferentia.

For Container Insights to collect AWS Neuron metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version `v1.5.0-eksbuild.1` or later.
- The Neuron driver must be installed on the nodes of the cluster.
- The Neuron device plugin must be installed on the cluster. For example, the Amazon EKS optimized accelerated AMIs are built with the necessary components.

The metrics that are collected are listed in the table in this section. The metrics are collected for AWS Trainium, AWS Inferentia, and AWS Inferentia2.

The CloudWatch agent collects these metrics from the Neuron monitor and does the necessary Kubernetes resource correlation to deliver metrics at the pod and container levels

Metric name	Dimensions	Description

`container_neuroncore_utilization`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName` , `NeuronDevice` , `NeuronCore`
		NeuronCore utilization, during the captured period of the NeuronCore allocated to the container.
		
		Unit: Percent

`container_neuroncore_memory_usage_constants`	`ClusterName`
	
	`ClusterName` , `Namespace` , `PodName`, `ContainerName`
	
	`ClusterName` , `Namespace` , `PodName`, `FullPodName` , `ContainerName`
		The amount of device memory used for constants during training by the NeuronCore that is allocated to the container (or weights during inference).
		
		Unit: Bytes

Metric name	Dimensions	Description
	ClusterName ,Namespace ,PodName, FullPodName , ContainerName ,NeuronDevice ,NeuronCore	
container_neuroncore_memory_usage_model_code	ClusterName ClusterName ,Namespace , PodName, ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ,NeuronDevice ,NeuronCore	The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the container. Unit: Bytes
container_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName ,Namespace , PodName, ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ClusterName ,Namespace ,PodName, FullPodName , ContainerName ,NeuronDevice ,NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the container. This memory region is reserved for the models. Unit: Bytes

Metric name	Dimensions	Description
container_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the container. Unit: Bytes
container_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The amount of device memory used for tensors by the NeuronCore allocated to the container. Unit: Bytes

Metric name	Dimensions	Description
container_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the container. Unit: Bytes
container_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

Metric name	Dimensions	Description
pod_neuroncore_utilization	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The NeuronCore utilization during the captured period of the NeuronCore allocated to the pod. Unit: Percent
pod_neuroncore_memory_usage_constants	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The amount of device memory used for constants during training by the NeuronCore that is allocated to the pod (or weights during inference). Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_model_code	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for the models' executable code by the NeuronCore that is allocated to the pod. Unit: Bytes
pod_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName ,Namespace ClusterName ,Namespace , Service ClusterName ,Namespace , PodName, FullPodName ClusterName ,Namespace , PodName, FullPodName , NeuronDevice ,NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the pod. This memory region is reserved for the models. Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore allocated to the pod. Unit: Bytes
pod_neuroncore_memory_usage_tensors	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The amount of device memory used for tensors by the NeuronCore allocated to the pod. Unit: Bytes

Metric name	Dimensions	Description
pod_neuroncore_memory_usage_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice , NeuronCore	The total amount of memory used by the NeuronCore allocated to the pod. Unit: Bytes
pod_neurondevice_hw_ecc_events_total	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device allocated to a pod. Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_utilization	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName ClusterName ,InstanceType ,InstanceId ,NodeName, NeuronDevice ,NeuronCore	The NeuronCore utilization during the captured period of the NeuronCore allocated to the node. Unit: Percent
node_neuroncore_memory_usage_constants	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName ClusterName ,InstanceType ,InstanceId ,NodeName, NeuronDevice ,NeuronCore	The amount of device memory used for constants during training by the NeuronCore that is allocated to the node (or weights during inference). Unit: Bytes
node_neuroncore_memory_usage_model_code	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName ClusterName ,InstanceType ,InstanceId ,NodeName, NeuronDevice ,NeuronCore	The amount of device memory used for models' executable code by the NeuronCore that is allocated to the node. Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_memory_usage_model_shared_scratchpad	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId ,NodeName ClusterName ,InstanceType ,InstanceId ,NodeName,NeuronDevice ,NeuronCore	The amount of device memory used for the scratchpad shared of the models by the NeuronCore that is allocated to the node. This is a memory region reserved for the models. Unit: Bytes
node_neuroncore_memory_usage_runtime_memory	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId ,NodeName ClusterName ,InstanceType ,InstanceId ,NodeName,NeuronDevice ,NeuronCore	The amount of device memory used for the Neuron runtime by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuroncore_memory_usage_tensors	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId ,NodeName ClusterName ,InstanceType ,InstanceId ,NodeName,NeuronDevice ,NeuronCore	The amount of device memory used for tensors by the NeuronCore that is allocated to the node. Unit: Bytes

Metric name	Dimensions	Description
node_neuroncore_memory_usage_total	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName ClusterName ,InstanceType ,InstanceId ,NodeName, NeuronDevice ,NeuronCore	The total amount of memory used by the NeuronCore that is allocated to the node. Unit: Bytes
node_neuron_execution_errors_total	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName	The total number of execution errors on the node. This is calculated by the CloudWatch agent by aggregating the errors of the following types: generic, numerical ,transient ,model, runtime, and hardware Unit: Count
node_neurondevice_runtime_memory_used_bytes	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName	The total Neuron device memory usage in bytes on the node. Unit: Bytes
node_neuron_execution_latency	ClusterName ClusterName ,UltraServer ClusterName ,InstanceId , NodeName	In seconds, the latency for an execution on the node as measured by the Neuron runtime. Unit: Seconds

Metric name	Dimensions	Description
node_neuron device_hw_ecc_events_total	ClusterName ClusterName , UltraServer ClusterName , InstanceId , NodeName ClusterName , InstanceId , NodeName, NeuronDevice	The number of corrected and uncorrected ECC events for the on-chip SRAM and device memory of the Neuron device on the node. Unit: Count

AWS Elastic Fabric Adapter (EFA) metrics

Beginning with version 1.300037.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS collects AWS Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters on Linux instances. The CloudWatch agent must be installed using the CloudWatch Observability EKS add-on version v1.5.2-eksbuild.1 or later. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about AWS Elastic Fabric Adapter, see Elastic Fabric Adapter.

For Container Insights to collect AWS Elastic Fabric adapter metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the Amazon CloudWatch Observability EKS add-on version v1.5.2-eksbuild.1 or later.
- The EFA device plugin must be installed on the cluster. For more information, see aws-efa-k8s-device-plugin on GitHub.

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description
container_efa_rx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName	The number of bytes per second received by the EFA device(s) allocated to the container.

Metric name	Dimensions	Description
	ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	Unit: Bytes/Second
container_efa_tx_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second transmitted by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_rx_dropped	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of packets that were received and then dropped by the EFA device(s) allocated to the container. Unit: Count/Second

Metric name	Dimensions	Description
container_efa_rdma_read_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second
container_efa_rdma_write_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the container. Unit: Bytes/Second

Metric name	Dimensions	Description
container_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace , PodName, ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName ClusterName , Namespace , PodName, FullPodName , ContainerName , NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the container. Unit: Bytes/Second
pod_efa_rx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received by the EFA device(s) allocated to the pod. Unit: Bytes/Second

Metric name	Dimensions	Description
pod_efa_tx_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second transmitted by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rx_dropped	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of packets that were received and then dropped by the EFA device(s) allocated to the pod. Unit: Count/Second

Metric name	Dimensions	Description
pod_efa_rdma_read_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
pod_efa_rdma_write_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second

Metric name	Dimensions	Description
pod_efa_rdma_write_recv_bytes	ClusterName ClusterName , Namespace ClusterName , Namespace , Service ClusterName , Namespace , PodName ClusterName , Namespace , PodName, FullPodName ClusterName , Namespace , PodName, FullPodName , NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second received by the EFA device(s) allocated to the node. Unit: Bytes/Second
node_efa_tx_bytes	ClusterName ClusterName , InstanceId , NodeName ClusterName , InstanceId , InstanceType , NodeName, NetworkInterfaceId	The number of bytes per second transmitted by the EFA device(s) allocated to the node. Unit: Bytes/Second

Metric name	Dimensions	Description
node_efa_rx_dropped	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, NetworkInterfaceId	The number of packets that were received and then dropped by the EFA device(s) allocated to the node. Unit: Count/Second
node_efa_rdma_read_bytes	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, NetworkInterfaceId	The number of bytes per second received using remote direct memory access read operations by the EFA device(s) allocated to the node. Unit: Bytes/Second
node_efa_rdma_write_bytes	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, NetworkInterfaceId	The number of bytes per second transmitted using remote direct memory access read operations by the EFA device(s) allocated to the pod. Unit: Bytes/Second
node_efa_rdma_write_recv_bytes	ClusterName ClusterName ,InstanceId , NodeName ClusterName ,InstanceId , InstanceType ,NodeName, NetworkInterfaceId	The number of bytes per second received during remote direct memory access write operations by the EFA device(s) allocated to the node. Unit: Bytes/Second

Amazon SageMaker AI HyperPod metrics

Beginning with version `v2.0.1-eksbuild.1` of the CloudWatch Observability EKS add-on, Container Insights with enhanced observability for Amazon EKS automatically collects Amazon SageMaker AI HyperPod metrics from Amazon EKS clusters. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about Amazon SageMaker AI HyperPod, see Amazon SageMaker AI HyperPod.

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description

`hyperpod_node_health_status_unschedulable`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `Unschedulable` by Amazon SageMaker AI HyperPod. This means that the node is running deep health checks and is not available for running workloads.
		
		Unit: Count

`hyperpod_node_health_status_schedulable`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `Schedulable` by Amazon SageMaker AI HyperPod. This means that the node has passed basic health checks or deep health checks and is available for running workloads.
		
		Unit: Count

`hyperpod_node_health_status_unschedulable_pen`	`ClusterName`
	
	`ClusterName` , `InstanceId` , `NodeName`
		Indicates if a node is labeled as `UnschedulablePendingReplacement` by HyperPod. This means that the node has failed deep health checks or health monitoring agent checks and requires a replacement.

Metric name	Dimensions	Description
ding_repl acement		If automatic node recovery is enabled, the node will be automatic ally replaced by Amazon SageMaker AI HyperPod. Unit: Count
hyperpod_ node_heal th_status _unschedu lable_pen ding_rebo ot	ClusterName ClusterName ,InstanceId , NodeName	Indicates if a node is labeled as UnschedulablePendi ngReboot by Amazon SageMaker AI HyperPod. This means that the node is running deep health checks and requires a reboot. If automatic node recovery is enabled, the node will be automatic ally rebooted by Amazon SageMaker AI HyperPod. Unit: Count

Amazon EBS NVMe driver metrics

Beginning with version 1.300056.0 of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS automatically collects Amazon EBS NVMe driver metrics from Amazon EKS clusters on Linux instances. The CloudWatch agent must be installed using the CloudWatch Observability Amazon EKS add-on version 4.1.0 or later. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about Amazon EBS, see Amazon EBS detailed performance statistics.

For Container Insights to collect Amazon EBS NVMe driver metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the CloudWatch Observability Amazon EKS add-on version 4.1.0 or later.

- The EBS CSI driver `1.42.0` add-on or Helm chart must be installed on the cluster with metrics enabled.
  - To enable the metrics when you are using Amazon EBS CSI driver add-on, use the following option when you create or update the add-on. `--configuration-values '{ "node": { "enableMetrics": true } }'`
  - To enable the metrics if you are using Helm chart, use the following option when you create or update the add-on. `--set node.enableMetrics=true`

The metrics that are collected are listed in the following table.

Metric name	Dimensions	Description

`node_diskio_ebs_total_read_ops`	`ClusterName`
	`ClusterName ,NodeName, InstanceId`
	`ClusterName ,NodeName, InstanceId VolumeId`
		The total number of completed read operations.

`node_diskio_ebs_total_write_ops`	`ClusterName`
	`ClusterName ,NodeName, InstanceId`
	`ClusterName ,NodeName, InstanceId VolumeId`
		The total number of completed write operations.

`node_diskio_ebs_total_read_bytes`	`ClusterName`
	`ClusterName ,NodeName, InstanceId`
	`ClusterName ,NodeName, InstanceId VolumeId`
		The total number of read bytes transferred.

`node_diskio_ebs_to`	`ClusterName`	The total number of write bytes transferred.

Metric name	Dimensions	Description
tal_write_bytes	ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	
node_diskio_ebs_total_read_time	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time spent, in microseconds, by all completed read operations.
node_diskio_ebs_total_write_time	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time spent, in microseconds, by all completed write operations.
node_diskio_ebs_volume_performance_exceeded_iops	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time, in microseconds, that IOPS demand exceeded the volume's provisioned IOPS performance.
node_diskio_ebs_volume_performance_exceeded_tp	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time, in microseconds, that throughput demand exceeded the volume's provisioned throughput performance.

Metric name	Dimensions	Description
node_diskio_ebs_ec2_instance_performance_exceeded_iops	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time, in microseconds, that the EBS volume exceeded the attached Amazon EC2 instance's maximum IOPS performance.
node_diskio_ebs_ec2_instance_performance_exceeded_tp	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The total time, in microseconds, that the EBS volume exceeded the attached Amazon EC2 instance's maximum throughput performance.
node_diskio_ebs_volume_queue_length	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId VolumeId	The number of read and write operations waiting to be completed.

Amazon EC2 local instance store NVMe metrics

Beginning with version `1.300069.0` of the CloudWatch agent, Container Insights with enhanced observability for Amazon EKS automatically collects Amazon EC2 local instance store NVMe metrics from Amazon EKS clusters on Linux instances. The CloudWatch agent must be installed using the CloudWatch Observability Amazon EKS add-on version `6.2.0` or later, or the equivalent Helm chart. For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart. For more information about Amazon EC2 instance store, see Amazon EC2 instance store.

To collect these metrics, you must meet the following prerequisites:

- You must be using Container Insights with enhanced observability for Amazon EKS, with the CloudWatch Observability Amazon EKS add-on version `6.2.0` or later.
- The `aws-ec2-local-instance-store-csi-driver` EKS add-on or Helm chart must be installed on the cluster with metrics enabled.
  - To enable the metrics when you are using the `aws-ec2-local-instance-store-csi-driver` add-on, use the following option when you create or update the add-on. `--configuration-values '{"metrics":{"enabled":true}}'`
  - To enable the metrics when you are using Helm chart, use the following option when you create or update the add-on. `--set metrics.enabled=true`
- Your cluster must have worker nodes running on EC2 instance types that support instance store volumes. For more information, see Considerations in the Amazon EKS User Guide.

After the prerequisites are met, the CloudWatch agent automatically discovers and collects metrics from the local instance store CSI driver. No additional CloudWatch agent configuration is required.

The following table lists the metrics that are collected.

Metric name	Dimensions	Description

`node_diskio_instance_store_total_read_ops`	`ClusterName`
	`ClusterName ,NodeName, InstanceId`
	`ClusterName ,NodeName, InstanceId ,VolumeId`
		The total number of completed read operations.

`node_diskio_instance_store_total_write_ops`	`ClusterName`
	`ClusterName ,NodeName, InstanceId`
	`ClusterName ,NodeName, InstanceId ,VolumeId`
		The total number of completed write operations.

`node_diskio_instance_store_`	`ClusterName`	The total number of read bytes transferred.

Metric name	Dimensions	Description
total_read_bytes	ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	
node_diskio_instance_store_total_write_bytes	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The total number of write bytes transferred.
node_diskio_instance_store_total_read_time	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The total time spent, in seconds, by all completed read operations.
node_diskio_instance_store_total_write_time	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The total time spent, in seconds, by all completed write operations.
node_diskio_instance_store_ec2_instance_performance_exceeded_iops	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The total time, in seconds, that the instance store volume exceeded the attached Amazon EC2 instance's maximum IOPS performance.

Metric name	Dimensions	Description
node_diskio_instance_store_ec2_instance_performance_exceeded_tp	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The total time, in seconds, that the instance store volume exceeded the attached Amazon EC2 instance's maximum throughput performance.
node_diskio_instance_store_volume_queue_length	ClusterName ClusterName ,NodeName, InstanceId ClusterName ,NodeName, InstanceId ,VolumeId	The number of read and write operations waiting to be completed.

Amazon EKS and Kubernetes Container Insights metrics

The following tables list the metrics and dimensions that Container Insights collects for Amazon EKS and Kubernetes. These metrics are in the `ContainerInsights` namespace. For more information, see Metrics.

If you do not see any Container Insights metrics in your console, be sure that you have completed the setup of Container Insights. Metrics do not appear before Container Insights has been set up completely. For more information, see Container Insights.

Metric name	Dimensions	Description
cluster_failed_node_count	ClusterName	The number of failed worker nodes in the cluster. A node is considered failed if it is suffering from any *node conditions*. For more information, see Conditions in the Kubernetes documentation.

Metric name	Dimensions	Description
cluster_node_count	ClusterName	The total number of worker nodes in the cluster.
namespace_number_of_running_pods	Namespace ClusterName ClusterName	The number of pods running per namespace in the resource that is specified by the dimensions that you're using.
node_cpu_limit	ClusterName	The maximum number of CPU units that can be assigned to a single node in this cluster.
node_cpu_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of CPU units that are reserved for node components, such as kubelet, kube-proxy, and Docker. Formula: node_cpu_request / node_cpu_limit **Note** node_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
node_cpu_usage_total	ClusterName	The number of CPU units being used on the nodes in the cluster.
node_cpu_utilization	NodeName, ClusterName , InstanceId ClusterName	The total percentage of CPU units being used on the nodes in the cluster. Formula: node_cpu_usage_total / node_cpu_limit
node_gpu_limit	ClusterName ClusterName ,InstanceId ,NodeName	The total number of GPU(s) available on the node.
node_gpu_usage_total	ClusterName ClusterName ,InstanceId ,NodeName	The number of GPU(s) being used by the running pods on the node.

Metric name	Dimensions	Description
node_gpu_reserved_capacity	ClusterName ClusterName , InstanceId , NodeName	The percentage of GPU currently being reserved on the node. The formula is, node_gpu_request / node_gpu_limit . **Note** node_gpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
node_filesystem_utilization	NodeName, ClusterName , InstanceId ClusterName	The total percentage of file system capacity being used on nodes in the cluster. Formula: node_filesystem_usage / node_filesystem_capacity **Note** node_filesystem_usage and node_filesystem_capacity are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_limit	ClusterName	The maximum amount of memory, in bytes, that can be assigned to a single node in this cluster.

Metric name	Dimensions	Description
node_memory_reserved_capacity	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being reserved on the nodes in the cluster. Formula: node_memory_request / node_memory_limit **Note** node_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_memory_utilization	NodeName, ClusterName , InstanceId ClusterName	The percentage of memory currently being used by the node or nodes. It is the percentage of node memory usage divided by the node memory limitation. Formula: node_memory_working_set / node_memory_limit .

Metric name	Dimensions	Description
node_memory_working_set	ClusterName	The amount of memory, in bytes, being used in the working set of the nodes in the cluster.
node_network_total_bytes	NodeName, ClusterName , InstanceId ClusterName	The total number of bytes per second transmitted and received over the network per node in a cluster. Formula: node_network_rx_bytes + node_network_tx_bytes **Note** node_network_rx_bytes and node_network_tx_bytes are not reported directly as metrics, but are fields in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
node_number_of_running_containers	NodeName, ClusterName , InstanceId ClusterName	The number of running containers per node in a cluster.

Metric name	Dimensions	Description
node_number_of_running_pods	NodeName, ClusterName , InstanceId ClusterName	The number of running pods per node in a cluster.
pod_cpu_reserved_capacity	PodName, Namespace , ClusterName ClusterName	The CPU capacity that is reserved per pod in a cluster. Formula: pod_cpu_request / node_cpu_limit **Note** pod_cpu_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_cpu_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of CPU units being used by pods. Formula: pod_cpu_usage_total / node_cpu_limit

Metric name	Dimensions	Description
pod_cpu_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of CPU units being used by pods relative to the pod limit. Formula: pod_cpu_usage_total / pod_cpu_limit
pod_gpu_request	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The GPU requests for the pod. This value must always be equal to pod_gpu_limit .
pod_gpu_limit	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The maximum number of GPU(s) that can be assigned to the pod in a node.
pod_gpu_usage_total	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The number of GPU(s) being allocated on the pod.

Metric name	Dimensions	Description
pod_gpu_reserved_capacity	ClusterName ClusterName , Namespace , PodName ClusterName , FullPodName , Namespace , PodName	The percentage of GPU currently being reserved for the pod. The formula is - pod_gpu_request / node_gpu_reserved_capacity.
pod_memory_reserved_capacity	PodName, Namespace , ClusterName ClusterName	The percentage of memory that is reserved for pods. Formula: pod_memory_request / node_memory_limit **Note** pod_memory_request is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_memory_utilization	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of memory currently being used by the pod or pods. Formula: pod_memory_working_set / node_memory_limit
pod_memory_utilization_over_pod_limit	PodName, Namespace , ClusterName Namespace , ClusterName Service, Namespace , ClusterName ClusterName	The percentage of memory that is being used by pods relative to the pod limit. If any containers in the pod don't have a memory limit defined, this metric doesn't appear. Formula: pod_memory_working_set / pod_memory_limit

Metric name	Dimensions	Description
pod_network_rx_bytes	PodName, Namespace , ClusterName Namespace ,ClusterName Service, Namespace , ClusterName ClusterName	The number of bytes per second being received over the network by the pod. Formula: sum(pod_interface_network_rx_bytes) **Note** pod_interface_network_rx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.

Metric name	Dimensions	Description
pod_network_tx_bytes	PodName, Namespace , ClusterName Namespace ,ClusterName Service, Namespace , ClusterName ClusterName	The number of bytes per second being transmitted over the network by the pod. Formula: sum(pod_interface_network_tx_bytes) **Note** pod_interface_network_tx_bytes is not reported directly as a metric, but is a field in performance log events. For more information, see Relevant fields in performance log events for Amazon EKS and Kubernetes.
pod_number_of_container_restarts	PodName, Namespace , ClusterName	The total number of container restarts in a pod.
service_number_of_running_pods	Service, Namespace , ClusterName ClusterName	The number of pods running the service or services in the cluster.

Kueue metrics

Beginning with version v2.4.0-eksbuild.1 of the the CloudWatch Observability EKS add-on, Container Insights for Amazon EKS supports collecting Kueue metrics from Amazon EKS clusters.

For more information about the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

For information about enabling the metrics, see Enable Kueue metrics to enable them.

The Kueue metrics that are collected are listed in the following table. These metrics are published into the `ContainerInsights/Prometheus` namespace in CloudWatch. Some of these metrics use the following dimensions:

- `ClusterQueue` is the name of the ClusterQueue
- The possible values of `Status` are `active` and `inadmissible`
- The possible values of `Reason` are `Preempted`, `PodsReadyTimeout`, `AdmissionCheck`, `ClusterQueueStopped`, and `InactiveWorkload`
- `Flavor` is the referenced flavor.
- `Resource` refers to cluster computer resources, such as `cpu`, `memory`, `gpu`, and so on.

Metric name	Dimensions	Description

`kueue_pending_workloads`	`ClusterName` , `ClusterQueue` , `Status`
	`ClusterName` , `ClusterQueue`
	`ClusterName` , `Status`
	`ClusterName`
		The number of pending workloads.

`kueue_evicted_workloads_total`	`ClusterName` , `ClusterQueue` , `Reason`
	`ClusterName` , `ClusterQueue`
	`ClusterName` , `Reason`
	`ClusterName`
		The total number of evicted workloads.

`kueue_admitted_act`	`ClusterName` , `ClusterQueue` `ClusterName`	The number of admitted workloads that are active (unsuspended and not finished).

Metric name	Dimensions	Description
ive_workloads		
kueue_cluster_queue_resource_usage	ClusterName ,ClusterQueue , Resource, Flavor ClusterName ,ClusterQueue , Resource ClusterName ,ClusterQueue , Flavor ClusterName ,ClusterQueue ClusterName	Reports the total resource usage of the ClusterQueue.
kueue_cluster_queue_nominal_quota	ClusterName ,ClusterQueue , Resource, Flavor ClusterName ,ClusterQueue , Resource ClusterName ,ClusterQueue , Flavor ClusterName ,ClusterQueue ClusterName	Reports the resource quota of the ClusterQueue.

Container Insights performance log reference

This section includes reference information about how Container Insights uses performance log events to collect metrics. When you deploy Container Insights, it automatically creates a log group for the performance log events. You don't need to create this log group yourself.

Topics

- Container Insights performance log events for Amazon ECS

- Container Insights performance log events for Amazon EKS and Kubernetes
- Relevant fields in performance log events for Amazon EKS and Kubernetes

Container Insights performance log events for Amazon ECS

The following are examples of the performance log events that Container Insights collects from Amazon ECS.

These logs are in CloudWatch Logs, in a log group named `/aws/ecs/containerinsights/`*`CLUSTER_NAME`*`/performance`. Within that log group, each container instance will has a log stream named `AgentTelemetry-`*`CONTAINER_INSTANCE_ID`*.

You can query these logs using queries such as `{ $.Type = "Container" }` to view all container log events.

Type: Container

```
{
 "Version":"0",
 "Type":"Container",
 "ContainerName":"sleep",
 "TaskId":"7ac4dfba69214411b4783a3b8189c9ba",
 "TaskDefinitionFamily":"sleep360",
 "TaskDefinitionRevision":"1",
 "ContainerInstanceId":"0d7650e6dec34c1a9200f72098071e8f",
 "EC2InstanceId":"i-0c470579dbcdbd2f3",
 "ClusterName":"MyCluster",
 "Image":"busybox",
 "ContainerKnownStatus":"RUNNING",
 "Timestamp":1623963900000,
 "CpuUtilized":0.0,
 "CpuReserved":10.0,
 "MemoryUtilized":0,
 "MemoryReserved":10,
 "StorageReadBytes":0,
 "StorageWriteBytes":0,
 "NetworkRxBytes":0,
 "NetworkRxDropped":0,
 "NetworkRxErrors":0,
 "NetworkRxPackets":14,
 "NetworkTxBytes":0,
 "NetworkTxDropped":0,
```

```
 "NetworkTxErrors":0,
 "NetworkTxPackets":0
}
```

Type: Task

Even though the units for `StorageReadBytes` and `StorageWriteBytes` are in Bytes/Second, the values represent the cumulative number of bytes read from and written to storage, respectively.

```
{
    "Version": "0",
    "Type": "Task",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "TaskDefinitionFamily": "sleep360",
    "TaskDefinitionRevision": "1",
    "ContainerInstanceId": "0d7650e6dec34c1a9200f72098071e8f",
    "EC2InstanceId": "i-0c470579dbcdbd2f3",
    "ClusterName": "MyCluster",
    "AccountID": "637146863587",
    "Region": "us-west-2",
    "AvailabilityZone": "us-west-2b",
    "KnownStatus": "RUNNING",
    "LaunchType": "EC2",
    "PullStartedAt": 1623963608201,
    "PullStoppedAt": 1623963610065,
    "CreatedAt": 1623963607094,
    "StartedAt": 1623963610382,
    "Timestamp": 1623963900000,
    "CpuUtilized": 0.0,
    "CpuReserved": 10.0,
    "MemoryUtilized": 0,
    "MemoryReserved": 10,
    "StorageReadBytes": 0,
    "StorageWriteBytes": 0,
    "NetworkRxBytes": 0,
    "NetworkRxDropped": 0,
    "NetworkRxErrors": 0,
    "NetworkRxPackets": 14,
    "NetworkTxBytes": 0,
    "NetworkTxDropped": 0,
    "NetworkTxErrors": 0,
    "NetworkTxPackets": 0,
```

```
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "CpuUtilized",
                    "Unit": "None"
                },
                {
                    "Name": "CpuReserved",
                    "Unit": "None"
                },
                {
                    "Name": "MemoryUtilized",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "MemoryReserved",
                    "Unit": "Megabytes"
                },
                {
                    "Name": "StorageReadBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "StorageWriteBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkRxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "NetworkTxBytes",
                    "Unit": "Bytes/Second"
                },
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
                    "Name": "EBSFilesystemUtilzed",
```

```
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "ClusterName",
                    "TaskDefinitionFamily"
                ]
            ]
        }
    ]
}
```

Type: Service

```
{
    "Version": "0",
    "Type": "Service",
    "ServiceName": "myCIService",
    "ClusterName": "myCICluster",
    "Timestamp": 1561586460000,
    "DesiredTaskCount": 2,
    "RunningTaskCount": 2,
    "PendingTaskCount": 0,
    "DeploymentCount": 1,
    "TaskSetCount": 0,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "DesiredTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "RunningTaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "PendingTaskCount",
                    "Unit": "Count"
                },
```

```
                {
                    "Name": "DeploymentCount",
                    "Unit": "Count"
                },
                {
                    "Name": "TaskSetCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Type: Volume

```
{
    "Version": "0",
    "Type": "Volume",
    "TaskDefinitionFamily": "myCITaskDef",
    "TaskId": "7ac4dfba69214411b4783a3b8189c9ba",
    "ClusterName": "myCICluster",
    "ServiceName": "myCIService",
    "VolumeId": "vol-1233436545ff708cb",
    "InstanceId": "i-0c470579dbcdbd2f3",
    "LaunchType": "EC2",
    "VolumeName": "MyVolumeName",
    "EBSFilesystemUtilized": 10,
    "EBSFilesystemSize": 20,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "EBSFilesystemSize",
                    "Unit": "Gigabytes"
                },
                {
```

```
                    "Name": "EBSFilesystemUtilzed",
                    "Unit": "Gigabytes"
                }
            ],
            "Dimensions": [
                ["ClusterName"],
                [
                    "VolumeName",
                    "TaskDefinitionFamily",
                    "ClusterName"
                ],
                [
                    "ServiceName",
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Type: Cluster

```
{
    "Version": "0",
    "Type": "Cluster",
    "ClusterName": "myCICluster",
    "Timestamp": 1561587300000,
    "TaskCount": 5,
    "ContainerInstanceCount": 5,
    "ServiceCount": 2,
    "CloudWatchMetrics": [
        {
            "Namespace": "ECS/ContainerInsights",
            "Metrics": [
                {
                    "Name": "TaskCount",
                    "Unit": "Count"
                },
                {
                    "Name": "ContainerInstanceCount",
                    "Unit": "Count"
                },
                {
```

```
                    "Name": "ServiceCount",
                    "Unit": "Count"
                }
            ],
            "Dimensions": [
                [
                    "ClusterName"
                ]
            ]
        }
    ]
}
```

Container Insights performance log events for Amazon EKS and Kubernetes

The following are examples of the performance log events that Container Insights collects from Amazon EKS and Kubernetes clusters.

Type: Node

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
```

```
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "node_network_total_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "node_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "node_memory_reserved_capacity"
        },
        {
          "Unit": "Count",
```

```
          "Name": "node_number_of_running_pods"
        },
        {
          "Unit": "Count",
          "Name": "node_number_of_running_containers"
        },
        {
          "Name": "node_cpu_usage_total"
        },
        {
          "Name": "node_cpu_limit"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_working_set"
        },
        {
          "Unit": "Bytes",
          "Name": "node_memory_limit"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "/proc",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567096682364",
  "Type": "Node",
  "Version": "0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
```

```
  },
  "node_cpu_limit": 4000,
  "node_cpu_request": 1130,
  "node_cpu_reserved_capacity": 28.249999999999996,
  "node_cpu_usage_system": 33.794636630852764,
  "node_cpu_usage_total": 136.47852169244098,
  "node_cpu_usage_user": 71.67075111567326,
  "node_cpu_utilization": 3.4119630423110245,
  "node_memory_cache": 3103297536,
  "node_memory_failcnt": 0,
  "node_memory_hierarchical_pgfault": 0,
  "node_memory_hierarchical_pgmajfault": 0,
  "node_memory_limit": 16624865280,
  "node_memory_mapped_file": 406646784,
  "node_memory_max_usage": 4230746112,
  "node_memory_pgfault": 0,
  "node_memory_pgmajfault": 0,
  "node_memory_request": 1115684864,
  "node_memory_reserved_capacity": 6.7109407818311055,
  "node_memory_rss": 798146560,
  "node_memory_swap": 0,
  "node_memory_usage": 3901444096,
  "node_memory_utilization": 6.601302600149552,
  "node_memory_working_set": 1097457664,
  "node_network_rx_bytes": 35918.392817386324,
  "node_network_rx_dropped": 0,
  "node_network_rx_errors": 0,
  "node_network_rx_packets": 157.67565245448117,
  "node_network_total_bytes": 68264.20276554905,
  "node_network_tx_bytes": 32345.80994816272,
  "node_network_tx_dropped": 0,
  "node_network_tx_errors": 0,
  "node_network_tx_packets": 154.21455923431654,
  "node_number_of_running_containers": 16,
  "node_number_of_running_pods": 13
}
```


Type: NodeFS


```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
```

```
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "node_filesystem_utilization"
        }
      ],
      "Dimensions": [
        [
          "NodeName",
          "InstanceId",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097939726",
  "Type": "NodeFS",
  "Version": "0",
  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_filesystem_available": 17298395136,
  "node_filesystem_capacity": 21462233088,
  "node_filesystem_inodes": 10484720,
  "node_filesystem_inodes_free": 10367158,
  "node_filesystem_usage": 4163837952,
  "node_filesystem_utilization": 19.400767547940255
```

```
}
```

Type: NodeDiskIO

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "Sources": [
    "cadvisor"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeDiskIO",
  "Version": "0",
  "device": "/dev/nvme0n1",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_diskio_io_service_bytes_async": 9750.505814277016,
  "node_diskio_io_service_bytes_read": 0,
  "node_diskio_io_service_bytes_sync": 230.6174506688036,
  "node_diskio_io_service_bytes_total": 9981.123264945818,
  "node_diskio_io_service_bytes_write": 9981.123264945818,
  "node_diskio_io_serviced_async": 1.153087253344018,
  "node_diskio_io_serviced_read": 0,
  "node_diskio_io_serviced_sync": 0.03603397666700056,
  "node_diskio_io_serviced_total": 1.1891212300110185,
  "node_diskio_io_serviced_write": 1.1891212300110185
}
```

Type: NodeNet

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
```

```
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567096928131",
  "Type": "NodeNet",
  "Version": "0",
  "interface": "eni972f6bfa9a0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal"
  },
  "node_interface_network_rx_bytes": 3163.008420864309,
  "node_interface_network_rx_dropped": 0,
  "node_interface_network_rx_errors": 0,
  "node_interface_network_rx_packets": 16.575629266820258,
  "node_interface_network_total_bytes": 3518.3935157426017,
  "node_interface_network_tx_bytes": 355.385094878293,
  "node_interface_network_tx_dropped": 0,
  "node_interface_network_tx_errors": 0,
  "node_interface_network_tx_packets": 3.9997714100370625
}
```

Type: Pod

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization"
        },
        {
          "Unit": "Bytes/Second",
          "Name": "pod_network_rx_bytes"
        },
        {
```

```
          "Unit": "Bytes/Second",
          "Name": "pod_network_tx_bytes"
        },
        {
          "Unit": "Percent",
          "Name": "pod_cpu_utilization_over_pod_limit"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_utilization_over_pod_limit"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Percent",
          "Name": "pod_cpu_reserved_capacity"
        },
        {
          "Unit": "Percent",
          "Name": "pod_memory_reserved_capacity"
        }
      ],
```

```
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    },
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "pod_number_of_container_restarts"
        }
      ],
      "Dimensions": [
        [
          "PodName",
          "Namespace",
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "Pod",
  "Version": "0",
```

```
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_cpu_limit": 200,
  "pod_cpu_request": 200,
  "pod_cpu_reserved_capacity": 5,
  "pod_cpu_usage_system": 1.4504841104992765,
  "pod_cpu_usage_total": 5.817016867430125,
  "pod_cpu_usage_user": 1.1281543081661038,
  "pod_cpu_utilization": 0.14542542168575312,
  "pod_cpu_utilization_over_pod_limit": 2.9085084337150624,
  "pod_memory_cache": 8192,
  "pod_memory_failcnt": 0,
  "pod_memory_hierarchical_pgfault": 0,
  "pod_memory_hierarchical_pgmajfault": 0,
  "pod_memory_limit": 104857600,
  "pod_memory_mapped_file": 0,
  "pod_memory_max_usage": 25268224,
  "pod_memory_pgfault": 0,
  "pod_memory_pgmajfault": 0,
  "pod_memory_request": 104857600,
  "pod_memory_reserved_capacity": 0.6307275170893897,
  "pod_memory_rss": 22777856,
  "pod_memory_swap": 0,
  "pod_memory_usage": 25141248,
  "pod_memory_utilization": 0.10988455961791709,
  "pod_memory_utilization_over_pod_limit": 17.421875,
  "pod_memory_working_set": 18268160,
  "pod_network_rx_bytes": 9880.697124714186,
  "pod_network_rx_dropped": 0,
```

```
  "pod_network_rx_errors": 0,
  "pod_network_rx_packets": 107.80005532263283,
  "pod_network_total_bytes": 10158.829201483635,
  "pod_network_tx_bytes": 278.13207676944796,
  "pod_network_tx_dropped": 0,
  "pod_network_tx_errors": 0,
  "pod_network_tx_packets": 1.146027574644318,
  "pod_number_of_container_restarts": 0,
  "pod_number_of_containers": 1,
  "pod_number_of_running_containers": 1,
  "pod_status": "Running"
}
```

Type: PodNet

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097351092",
  "Type": "PodNet",
  "Version": "0",
  "interface": "eth0",
  "kubernetes": {
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
```

```
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "pod_interface_network_rx_bytes": 9880.697124714186,
  "pod_interface_network_rx_dropped": 0,
  "pod_interface_network_rx_errors": 0,
  "pod_interface_network_rx_packets": 107.80005532263283,
  "pod_interface_network_total_bytes": 10158.829201483635,
  "pod_interface_network_tx_bytes": 278.13207676944796,
  "pod_interface_network_tx_dropped": 0,
  "pod_interface_network_tx_errors": 0,
  "pod_interface_network_tx_packets": 1.146027574644318
}
```

Type: Container

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-NodeGroup-
sample",
  "ClusterName": "myCICluster",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "pod",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "Container",
  "Version": "0",
  "container_cpu_limit": 200,
  "container_cpu_request": 200,
  "container_cpu_usage_system": 1.87958283771964,
  "container_cpu_usage_total": 6.159993652997942,
  "container_cpu_usage_user": 1.6707403001952357,
```

```
  "container_cpu_utilization": 0.15399984132494854,
  "container_memory_cache": 8192,
  "container_memory_failcnt": 0,
  "container_memory_hierarchical_pgfault": 0,
  "container_memory_hierarchical_pgmajfault": 0,
  "container_memory_limit": 104857600,
  "container_memory_mapped_file": 0,
  "container_memory_max_usage": 24580096,
  "container_memory_pgfault": 0,
  "container_memory_pgmajfault": 0,
  "container_memory_request": 104857600,
  "container_memory_rss": 22736896,
  "container_memory_swap": 0,
  "container_memory_usage": 24453120,
  "container_memory_utilization": 0.10574541028701798,
  "container_memory_working_set": 17580032,
  "container_status": "Running",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  },
  "number_of_container_restarts": 0
}
```


Type: ContainerFS

```
{
  "AutoScalingGroupName": "eksctl-myCICluster-nodegroup-standard-workers-
NodeGroup-1174PV2WHZAYU",
  "ClusterName": "myCICluster",
  "EBSVolumeId": "aws://us-west-2b/vol-0a53108976d4a2fda",
  "InstanceId": "i-1234567890123456",
  "InstanceType": "t3.xlarge",
  "Namespace": "amazon-cloudwatch",
  "NodeName": "ip-192-0-2-0.us-west-2.compute.internal",
  "PodName": "cloudwatch-agent-statsd",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "cadvisor",
    "calculated"
  ],
  "Timestamp": "1567097399912",
  "Type": "ContainerFS",
  "Version": "0",

  "device": "/dev/nvme0n1p1",
  "fstype": "vfs",
  "kubernetes": {
    "container_name": "cloudwatch-agent",
    "docker": {
      "container_id":
 "8967b6b37da239dfad197c9fdea3e5dfd35a8a759ec86e2e4c3f7b401e232706"
    },
    "host": "ip-192-168-75-26.us-west-2.compute.internal",
    "labels": {
      "app": "cloudwatch-agent-statsd",
      "pod-template-hash": "df44f855f"
    },
    "namespace_name": "amazon-cloudwatch",
    "pod_id": "2f4ff5ac-c813-11e9-a31d-06e9dde32928",
    "pod_name": "cloudwatch-agent-statsd-df44f855f-ts4q2",
    "pod_owners": [
      {
        "owner_kind": "Deployment",
        "owner_name": "cloudwatch-agent-statsd"
      }
    ],
    "service_name": "cloudwatch-agent-statsd"
  }
```

```
}
```

Type: Cluster

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "cluster_node_count"
        },
        {
          "Unit": "Count",
          "Name": "cluster_failed_node_count"
        }
      ],
      "Dimensions": [
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "Cluster",
  "Version": "0",
  "cluster_failed_node_count": 0,
  "cluster_node_count": 3
}
```

Type: ClusterService

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
```

```
          "Unit": "Count",
          "Name": "service_number_of_running_pods"
        }
      ],
      "Dimensions": [
        [
          "Service",
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Service": "cloudwatch-agent-statsd",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097534160",
  "Type": "ClusterService",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch",
    "service_name": "cloudwatch-agent-statsd"
  },
  "service_number_of_running_pods": 1
}
```

Type: ClusterNamespace

```
{
  "CloudWatchMetrics": [
    {
      "Metrics": [
        {
          "Unit": "Count",
          "Name": "namespace_number_of_running_pods"
        }
```

```
      ],
      "Dimensions": [
        [
          "Namespace",
          "ClusterName"
        ],
        [
          "ClusterName"
        ]
      ],
      "Namespace": "ContainerInsights"
    }
  ],
  "ClusterName": "myCICluster",
  "Namespace": "amazon-cloudwatch",
  "Sources": [
    "apiserver"
  ],
  "Timestamp": "1567097594160",
  "Type": "ClusterNamespace",
  "Version": "0",
  "kubernetes": {
    "namespace_name": "amazon-cloudwatch"
  },
  "namespace_number_of_running_pods": 7
}
```


Relevant fields in performance log events for Amazon EKS and Kubernetes

For Amazon EKS and Kubernetes, the containerized CloudWatch agent emits data as performance log events. This enables CloudWatch to ingest and store high-cardinality data. CloudWatch uses the data in the performance log events to create aggregated CloudWatch metrics at the cluster, node, and pod levels without the need to lose granular details.

The following table lists the fields in these performance log events that are relevant to the collection of Container Insights metric data. You can use CloudWatch Logs Insights to query for any of these fields to collect data or investigate issues. For more information, see Analyze Log Data With CloudWatch Logs Insights.

Type	Log field	Source	Formula or notes
Pod	`pod_cpu_utilization`	Calculated	Formula: `pod_cpu_usage_total / node_cpu_limit`
Pod	`pod_cpu_usage_total` `pod_cpu_usage_total` is reported in millicores.	cadvisor	
Pod	`pod_cpu_limit`	Calculated	Formula: `sum(container_cpu_limit)` `sum(container_cpu_limit)` includes already-completed pods. If any containers in the pod don't have a CPU limit defined, this field doesn't appear in the log event. This

Type	Log field	Source	Formula or notes
			includes init containers.
Pod	pod_cpu_request	Calculated	Formula: sum(container_cpu_request) container_cpu_request isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	pod_cpu_utilization_over_pod_limit	Calculated	Formula: pod_cpu_usage_total / pod_cpu_limit
Pod	pod_cpu_reserved_capacity	Calculated	Formula: pod_cpu_request / node_cpu_limit

Type	Log field	Source	Formula or notes
Pod	pod_memory_utilization	Calculated	Formula: pod_memory_working_set / node_memory_limit It is the percentage of pod memory usage over the node memory limitation.
Pod	pod_memory_working_set	cadvisor	
Pod	pod_memory_limit	Calculated	Formula: sum(container_memory_limit) If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes init containers.

Type	Log field	Source	Formula or notes
Pod	`pod_memory_request`	Calculated	Formula: `sum(container_memory_request)` `container_memory_request` isn't guaranteed to be set. Only the ones that are set are included in the sum.
Pod	`pod_memory_utilization_over_pod_limit`	Calculated	Formula: `pod_memory_working_set / pod_memory_limit` If any containers in the pod don't have a memory limit defined, this field doesn't appear in the log event. This includes init containers.

Type	Log field	Source	Formula or notes
Pod	pod_memory_reserved_capacity	Calculated	Formula: pod_memory_request / node_memory_limit
Pod	pod_network_tx_bytes	Calculated	Formula: sum(pod_interface_network_tx_bytes) This data is available for all the network interfaces per pod. The CloudWatch agent calculates the total and adds metric extraction rules.
Pod	pod_network_rx_bytes	Calculated	Formula: sum(pod_interface_network_rx_bytes)

Type	Log field	Source	Formula or notes
Pod	pod_network_total_bytes	Calculated	Formula: pod_network_rx_bytes + pod_network_tx_bytes
PodNet	pod_interface_network_rx_bytes	cadvisor	This data is network rx bytes per second of a pod network interface.
PodNet	pod_interface_network_tx_bytes	cadvisor	This data is network tx bytes per second of a pod network interface.
Container	container_cpu_usage_total	cadvisor	
Container	container_cpu_limit	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_cpu_request	cadvisor	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_working_set	cadvisor	

Type	Log field	Source	Formula or notes
Container	container_memory_limit	pod	Not guaranteed to be set. It's not emitted if it's not set.
Container	container_memory_request	pod	Not guaranteed to be set. It's not emitted if it's not set.
Node	node_cpu_utilization	Calculated	Formula: node_cpu_usage_total / node_cpu_limit
Node	node_cpu_usage_total	cadvisor	
Node	node_cpu_limit	/proc	

Type	Log field	Source	Formula or notes
Node	node_cpu_request	Calculated	Formula: sum(pod_cpu_request) For cronjobs, node_cpu_request also includes requests from completed pods. This can lead to a high value for node_cpu_reserved_capacity .
Node	node_cpu_reserved_capacity	Calculated	Formula: node_cpu_request / node_cpu_limit
Node	node_memory_utilization	Calculated	Formula: node_memory_working_set / node_memory_limit
Node	node_memory_working_set	cadvisor	
Node	node_memory_limit	/proc	

Type	Log field	Source	Formula or notes
Node	node_memory_request	Calculated	Formula: sum(pod_memory_request)
Node	node_memory_reserved_capacity	Calculated	Formula: node_memory_request / node_memory_limit
Node	node_network_rx_bytes	Calculated	Formula: sum(node_interface_network_rx_bytes)
Node	node_network_tx_bytes	Calculated	Formula: sum(node_interface_network_tx_bytes)
Node	node_network_total_bytes	Calculated	Formula: node_network_rx_bytes + node_network_tx_bytes
Node	node_number_of_running_pods	Pod List	

Type	Log field	Source	Formula or notes
Node	`node_number_of_running_containers`	Pod List	
NodeNet	`node_interface_network_rx_bytes`	cadvisor	This data is network rx bytes per second of a worker node network interface.
NodeNet	`node_interface_network_tx_bytes`	cadvisor	This data is network tx bytes per second of a worker node network interface.
NodeFS	`node_filesystem_capacity`	cadvisor	
NodeFS	`node_filesystem_usage`	cadvisor	
NodeFS	`node_filesystem_utilization`	Calculated	Formula: `node_filesystem_usage / node_filesystem_capacity` This data is available per device name.

Type	Log field	Source	Formula or notes
Cluster	cluster_failed_node_count	API Server	
Cluster	cluster_node_count	API Server	
Service	service_number_of_running_pods	API Server	
Namespace	namespace_number_of_running_pods	API Server	

Metrics calculation examples

This section includes examples that show how some of the values in the preceding table are calculated.

Suppose that you have a cluster in the following state.

```
Node1
   node_cpu_limit = 4
   node_cpu_usage_total = 3

   Pod1
     pod_cpu_usage_total = 2

     Container1
        container_cpu_limit = 1
        container_cpu_request = 1
        container_cpu_usage_total = 0.8

     Container2
        container_cpu_limit = null
        container_cpu_request = null
        container_cpu_usage_total = 1.2

   Pod2
     pod_cpu_usage_total = 0.4

     Container3
```

```
        container_cpu_limit = 1
        container_cpu_request = 0.5
        container_cpu_usage_total = 0.4

Node2
   node_cpu_limit = 8
   node_cpu_usage_total = 1.5

   Pod3
     pod_cpu_usage_total = 1

     Container4
        container_cpu_limit = 2
        container_cpu_request = 2
        container_cpu_usage_total = 1
```

The following table shows how pod CPU metrics are calculated using this data.

Metric	Formula	Pod1	Pod2	Pod3
pod_cpu_utilization	pod_cpu_usage_total / node_cpu_limit	2 / 4 = 50%	0.4 / 4 = 10%	1 / 8 = 12.5%
pod_cpu_utilization_over_pod_limit	pod_cpu_usage_total / sum(container_cpu_limit)	N/A because CPU limit for Container 2 isn't defined	0.4 / 1 = 40%	1 / 2 = 50%
pod_cpu_reserved_capacity	sum(container_cpu_request) / node_cpu_limit	(1 + 0) / 4 = 25%	0.5 / 4 = 12.5%	2 / 8 = 25%

The following table shows how node CPU metrics are calculated using this data.

Metric	Formula	Node1	Node2
node_cpu_utilization	node_cpu_usage_total / node_cpu_limit	3 / 4 = 75%	1.5 / 8 = 18.75%
node_cpu_reserved_capacity	sum(pod_cpu_request) / node_cpu_limit	1.5 / 4 = 37.5%	2 / 8 = 25%

Container Insights Prometheus metrics monitoring

CloudWatch Container Insights monitoring for Prometheus automates the discovery of Prometheus metrics from containerized systems and workloads. Prometheus is an open-source systems monitoring and alerting toolkit. For more information, see What is Prometheus? in the Prometheus documentation.

Discovering Prometheus metrics is supported for Amazon Elastic Container Service, Amazon Elastic Kubernetes Service and Kubernetes clusters running on Amazon EC2 instances. The Prometheus counter, gauge, and summary metric types are collected.

For Amazon ECS and Amazon EKS clusters, both the EC2 and Fargate launch types are supported. Container Insights automatically collects metrics from several workloads, and you can configure it to collect metrics from any workload.

You can adopt Prometheus as an open-source and open-standard method to ingest custom metrics in CloudWatch. The CloudWatch agent with Prometheus support discovers and collects Prometheus metrics to monitor, troubleshoot, and alarm on application performance degradation and failures faster. This also reduces the number of monitoring tools required to improve observability.

Container Insights Prometheus support involves pay-per-use of metrics and logs, including collecting, storing, and analyzing. For more information, see Amazon CloudWatch Pricing.

Pre-built dashboards for some workloads

The Container Insights Prometheus solution includes pre-built dashboards for the popular workloads that are listed in this section. For sample configurations for these workloads, see (Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing and (Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing.

You can also configure Container Insights to collect Prometheus metrics from other containerized services and applications by editing the agent configuration file.

Workloads with pre-built dashboards for Amazon EKS clusters and Kubernetes clusters running on Amazon EC2 instances:

- AWS App Mesh
- NGINX
- Memcached
- Java/JMX
- HAProxy

Workloads with pre-built dashboards for Amazon ECS clusters:

- AWS App Mesh
- Java/JMX
- NGINX
- NGINX Plus

Set up and configure Prometheus metrics collection on Amazon ECS clusters

To collect Prometheus metrics from Amazon ECS clusters, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see https://aws-otel.github.io/docs/getting-started/container-insights/ecs-prometheus.

The following sections explain how to use the CloudWatch agent as the collector to retrieve Prometheus metrics. You install the CloudWatch agent with Prometheus monitoring on clusters running Amazon ECS, and you can optionally configure the agent to scrape additional targets. These sections also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Container Insights on Amazon ECS supports the following launch type and network mode combinations for Prometheus metrics:

Amazon ECS launch type	Network modes supported
EC2 (Linux)	bridge, host, and awsvpc
Fargate	awsvpc

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters
- Scraping additional Prometheus sources and importing those metrics
- (Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

Install the CloudWatch agent with Prometheus metrics collection on Amazon ECS clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon ECS. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- Java/JMX

You can also configure the agent to scrape and import metrics from additional Prometheus workloads and sources.

Set up IAM roles

You need two IAM roles for the CloudWatch agent task definition. If you specify **`CreateIAMRoles=True`** in the CloudFormation stack to have Container Insights create these roles for you, the roles will be created with the correct permissions. If you want to create them yourself or use existing roles, the following roles and permissions are required.

- **CloudWatch agent ECS task role**— The CloudWatch agent container uses this role. It must include the **CloudWatchAgentServerPolicy** policy and a customer-managed policy which contains the following read-only permissions:
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:ListServices`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeServices`
  - `ecs:DescribeTasks`
  - `ecs:DescribeTaskDefinition`
- **CloudWatch agent ECS task execution role**— This is the role that Amazon ECS requires to launch and execute your containers. Make sure that your task execution role has the **AmazonSSMReadOnlyAccess**, **AmazonECSTaskExecutionRolePolicy**, and **CloudWatchAgentServerPolicy** policies attached. If you want to store more sensitive data for Amazon ECS to use, see Specifying sensitive data.

Install the CloudWatch agent with Prometheus monitoring by using CloudFormation

You use AWS CloudFormation to install the CloudWatch agent with Prometheus monitoring for Amazon ECS clusters. The following list shows the parameters you will use in the CloudFormation template.

- **ECSClusterName**— Specifies the target Amazon ECS cluster.
- **CreateIAMRoles**— Specify **`True`** to create new roles for the Amazon ECS task role and Amazon ECS task execution role. Specify **`False`** to reuse existing roles.
- **TaskRoleName**— If you specified **`True`** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task role. If you specified **`False`** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task role.
- **ExecutionRoleName**— If you specified **`True`** for **CreateIAMRoles**, this specifies the name to use for the new Amazon ECS task execution role. If you specified **`False`** for **CreateIAMRoles**, this specifies the existing role to use as the Amazon ECS task execution role.
- **ECSNetworkMode**— If you are using EC2 launch type, specify the network mode here. It must be either **`bridge`** or **`host`**.
- **ECSLaunchType**— Specify either **`fargate`** or **`EC2`**.

- **SecurityGroupID**— If the **ECSNetworkMode** is **awsvpc**, specify the security group ID here.
- **SubnetID**— If the **ECSNetworkMode** is **awsvpc**, specify the subnet ID here.

Command samples

This section includes sample CloudFormation commands to install Container Insights with Prometheus monitoring in various scenarios.

Create CloudFormation stack for an Amazon ECS cluster in bridge network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=bridge
export CREATE_IAM_ROLES=True
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create CloudFormation stack for an Amazon ECS cluster in host network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_NETWORK_MODE=host
export CREATE_IAM_ROLES=True
```

```
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name


curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                 ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create CloudFormation stack for an Amazon ECS cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=EC2
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                 ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
```

```
                ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

Create CloudFormation stack for a Fargate cluster in awsvpc network mode

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
export ECS_CLUSTER_NAME=your_ec2_ecs_cluster_name
export ECS_LAUNCH_TYPE=FARGATE
export CREATE_IAM_ROLES=True
export ECS_CLUSTER_SECURITY_GROUP=your_security_group_eg_sg-xxxxxxxxxx
export ECS_CLUSTER_SUBNET=your_subnet_eg_subnet-xxxxxxxxxx
export ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
export ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-
prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-${ECS_LAUNCH_TYPE}-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                ParameterKey=CreateIAMRoles,ParameterValue=${CREATE_IAM_ROLES} \
                ParameterKey=ECSLaunchType,ParameterValue=${ECS_LAUNCH_TYPE} \
                ParameterKey=SecurityGroupID,ParameterValue=
${ECS_CLUSTER_SECURITY_GROUP} \
                ParameterKey=SubnetID,ParameterValue=${ECS_CLUSTER_SUBNET} \
                ParameterKey=TaskRoleName,ParameterValue=${ECS_TASK_ROLE_NAME} \
                ParameterKey=ExecutionRoleName,ParameterValue=
${ECS_EXECUTION_ROLE_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

AWS resources created by the CloudFormation stack

The following table lists the AWS resources that are created when you use CloudFormation to set up Container Insights with Prometheus monitoring on an Amazon ECS cluster.

Resource type	Resource name	Comments
AWS::SSM::Parameter	AmazonCloudWatch-CWAgentConfig-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	This is the CloudWatch agent with the default App Mesh and Java/JMX embedded metric format definition.
AWS::SSM::Parameter	AmazonCloudWatch-Prometheus ConfigName-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	This is the Prometheus scraping configuration.
AWS::IAM::Role	**$ECS_TASK_ROLE_NAME**.	The Amazon ECS task role. This is created only if you specified **`True`** for `CREATE_IAM_ROLES` .
AWS::IAM::Role	**${ECS_EXECUTION_ROLE_NAME}**	The Amazon ECS task execution role. This is created only if you specified **`True`** for `CREATE_IAM_ROLES` .
AWS::ECS::TaskDefinition	cwagent-prometheus-$*ECS_CLUSTER_NAME* -$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	
AWS::ECS::Service	cwagent-prometheus-replica-service-$*ECS_LAUNCH_TYPE* -$*ECS_NETWORK_MODE*	

Deleting the CloudFormation stack for the CloudWatch agent with Prometheus monitoring

To delete the CloudWatch agent from an Amazon ECS cluster, enter these commands.

```
export AWS_PROFILE=your_aws_config_profile_eg_default
export AWS_DEFAULT_REGION=your_aws_region_eg_ap-southeast-1
```

```
export CLOUDFORMATION_STACK_NAME=your_cloudformation_stack_name

aws cloudformation delete-stack \
--stack-name ${CLOUDFORMATION_STACK_NAME} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config> in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon ECS clusters, the configurations are integrated with the Parameter Store of AWS Systems Manager by the secrets in the Amazon ECS task definition:

- The secret `PROMETHEUS_CONFIG_CONTENT` is for the Prometheus scrape configuration.
- The secret `CW_CONFIG_CONTENT` is for the CloudWatch agent configuration.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
  scrape_interval: 1m
```

```
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in the YAML files for Amazon ECS such as `cwagent-ecs-prometheus-metric-for-bridge-host.yaml`, the default scraping jobs are configured in the `ecs_service_discovery` section.

```
"ecs_service_discovery": {
                  "sd_frequency": "1m",
                  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
                  "docker_label": {
                  },
                  "task_definition_list": [
                    {
                      "sd_job_name": "ecs-appmesh-colors",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition\/.*-
ColorTeller-(white):[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    },
                    {
                      "sd_job_name": "ecs-appmesh-gateway",
                      "sd_metrics_ports": "9901",
                      "sd_task_definition_arn_pattern": ".*:task-definition/.*-
ColorGateway:[0-9]+",
                      "sd_metrics_path": "/stats/prometheus"
                    }
                  ]
                }
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see Embedding metrics within logs.

Log events from Amazon ECS clusters are stored in the **/aws/ecs/containerinsights/*cluster_name*/prometheus** log group.

Each scraping job is contained in a different log stream in this log group.

To add a new scraping target, you add a new entry in the `task_definition_list` section under the `ecs_service_discovery` section. of the YAML file, and restart the agent. For an example of this process, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon ECS cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses **/aws/ecs/containerinsights/*cluster_name*/prometheus** for logs from Amazon ECS clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configurations of the Amazon ECS Prometheus target auto-discovery functions. Two modes are supported to discover the Prometheus targets: discovery based on the container's Docker label or discovery based on the Amazon ECS task definition ARN regular expression. You can use the two modes together and the CloudWatch agent will de-duplicate the discovered targets based on: *{private_ip}:{port}/{metrics_path}*.

  The `ecs_service_discovery` section can contain the following fields:

  - `sd_frequency` is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

    This field is optional. The default is 60 seconds (1 minute).

- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for Docker label-based service discovery. If you omit this section, Docker label-based discovery is not used. This section can contain the following fields:
  - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this Docker label, the CloudWatch agent will skip it.
  - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this Docker label, the agent assumes the default path `/metrics`.
  - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
  - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
  - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
  - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.

- `service_name_list_for_tasks` is an optional section that you can use to specify the configuration of service name-based discovery. If you omit this section, service name-based discovery is not used. This section can contain the following fields:
  - `sd_service_name_pattern` is the pattern to use to specify the Amazon ECS service where tasks are to be discovered. This is a regular expression.
  - `sd_metrics_ports` Lists the `containerPort` for the Prometheus metrics. Separate multiple `containerPorts` with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.
  - `sd_metrics_path` specifies the Prometheus metrics path. If you omit this, the agent assumes that the default path `/metrics`.
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
  - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

    If you have multiple labels specified in `source_labels`, we recommend that you do not use `^` or `$` characters in the regular expression for `label_matcher`.
  - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
  - `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
  - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
  - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system$",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
```

```
    "Service":"kube-dns",
    "coredns_dns_request_type_count_total":2562,
    "eks_amazonaws_com_component":"kube-dns",
    "instance":"192.168.61.254:9153",
    "job":"kubernetes-service-endpoints",
    ...
}
```

Detailed guide for autodiscovery on Amazon ECS clusters

Prometheus provides dozens of dynamic service-discovery mechanisms as described in <scrape_config>. However there is no built-in service discovery for Amazon ECS. The CloudWatch agent adds this mechanism.

When the Amazon ECS Prometheus service discovery is enabled, the CloudWatch agent periodically makes the following API calls to Amazon ECS and Amazon EC2 frontends to retrieve the metadata of the running ECS tasks in the target ECS cluster.

```
EC2:DescribeInstances
ECS:ListTasks
ECS:ListServices
ECS:DescribeContainerInstances
ECS:DescribeServices
ECS:DescribeTasks
ECS:DescribeTaskDefinition
```

The metadata is used by the CloudWatch agent to scan the Prometheus targets within the ECS cluster. The CloudWatch agent supports three service discovery modes:

- Container Docker label-based service discovery
- ECS task definition ARN regular expression-based service discovery
- ECS service name regular expression-based service discovery

All modes can be used together. CloudWatch agent de-duplicates the discovered targets based on: `{private_ip}:{port}/{metrics_path}`.

All discovered targets are written into a result file specified by the `sd_result_file` configuration field within the CloudWatch agent container. The following is a sample result file:

```
- targets:
```

```
  - 10.6.1.95:32785
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT: "9406"
    ECS_PROMETHEUS_JOB_NAME: demo-jar-ec2-bridge-dynamic
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-jar-ec2-bridge-dynamic-port
    TaskGroup: family:demo-jar-ec2-bridge-dynamic-port
    TaskRevision: "7"
    VpcId: vpc-01234567890
    container_name: demo-jar-ec2-bridge-dynamic-port
    job: demo-jar-ec2-bridge-dynamic
- targets:
  - 10.6.3.193:9404
  labels:
    __metrics_path__: /metrics
    ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_B: "9404"
    ECS_PROMETHEUS_JOB_NAME: demo-tomcat-ec2-bridge-mapped-port
    ECS_PROMETHEUS_METRICS_PATH: /metrics
    InstanceType: t3.medium
    LaunchType: EC2
    SubnetId: subnet-123456789012
    TaskDefinitionFamily: demo-tomcat-ec2-bridge-mapped-port
    TaskGroup: family:demo-jar-tomcat-bridge-mapped-port
    TaskRevision: "12"
    VpcId: vpc-01234567890
    container_name: demo-tomcat-ec2-bridge-mapped-port
    job: demo-tomcat-ec2-bridge-mapped-port
```

You can directly integrate this result file with Prometheus file-based service discovery. For more information about Prometheus file-based service discovery, see <file_sd_config>.

Suppose the result file is written to `/tmp/cwagent_ecs_auto_sd.yaml` The following Prometheus scrape configuration will consume it.

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: cwagent-ecs-file-sd-config
```

```
    sample_limit: 10000
    file_sd_configs:
      - files: [ "/tmp/cwagent_ecs_auto_sd.yaml" ]
```

The CloudWatch agent also adds the following additional labels for the discovered targets.

- `container_name`
- `TaskDefinitionFamily`
- `TaskRevision`
- `TaskGroup`
- `StartedBy`
- `LaunchType`
- `job`
- `__metrics_path__`
- Docker labels

When the cluster has the EC2 launch type, the following three labels are added.

- `InstanceType`
- `VpcId`
- `SubnetId`

> **Note**
>
> Docker labels that don't match the regular expression `[a-zA-Z_][a-zA-Z0-9_]*` are filtered out. This matches the Prometheus conventions as listed in `label_name` in Configuration file in the Prometheus documentation.

ECS service discovery configuration examples

This section includes examples that demonstrate ECS service discovery.

Example 1

```
"ecs_service_discovery": {
```

```
  "sd_frequency": "1m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
  }
}
```

This example enables Docker label-based service discovery. The CloudWatch agent will query the ECS tasks' metadata once per minute and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

The default value of `sd_port_label` in the `docker_label` section is `ECS_PROMETHEUS_EXPORTER_PORT`. If any running container in the ECS tasks has a `ECS_PROMETHEUS_EXPORTER_PORT` docker label, the CloudWatch agent uses its value as `container port` to scan all exposed ports of the container. If there is a match, the mapped host port plus the private IP of the container are used to construct the Prometheus exporter target in the following format: `private_ip:host_port`.

The default value of `sd_metrics_path_label` in the `docker_label` section is `ECS_PROMETHEUS_METRICS_PATH`. If the container has this Docker label, its value will be used as the `__metrics_path__` . If the container does not have this label, the default value `/metrics` is used.

The default value of `sd_job_name_label` in the `docker_label` section is `job`. If the container has this Docker label, its value will be appended as one of the labels for the target to replace the default job name specified in the Prometheus configuration. The value of this Docker label is used as the log stream name in the CloudWatch Logs log group.

Example 2

```
"ecs_service_discovery": {
  "sd_frequency": "15s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A",
    "sd_job_name_label": "ECS_PROMETHEUS_JOB_NAME"
  }
}
```

This example enables Docker label-based service discovery. THe CloudWatch agent will query the ECS tasks' metadata every 15 seconds and write the discovered targets into the `/tmp/`

cwagent_ecs_auto_sd.yaml file within the CloudWatch agent container. The containers with a Docker label of ECS_PROMETHEUS_EXPORTER_PORT_SUBSET_A will be scanned. The value of the docker label ECS_PROMETHEUS_JOB_NAME is used as the job name.

Example 3

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "task_definition_list": [
    {
      "sd_job_name": "java-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9404; 9406",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*javajmx.*:[0-9]+"
    },
    {
      "sd_job_name": "envoy-prometheus",
      "sd_metrics_path": "/stats/prometheus",
      "sd_container_name_pattern": "^envoy$",
      "sd_metrics_ports": "9901",
      "sd_task_definition_arn_pattern": ".*:task-definition/.*appmesh.*:23"
    }
  ]
}
```

This example enables ECS task definition ARN regular expression-based service discovery. The CloudWatch agent will query the ECS tasks' metadata every five minutes and write the discovered targets into the /tmp/cwagent_ecs_auto_sd.yaml file within the CloudWatch agent container.

Two task definition ARN regular expresion sections are defined:

- For the first section, the ECS tasks with javajmx in their ECS task definition ARN are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of sd_metrics_path sets __metrics_path__ to /metrics. So the CloudWatch agent will scrape the Prometheus metrics from private_ip:host_port/metrics, the scraped metrics are sent to the java-prometheus log stream in CloudWatch Logs in the log group /aws/ecs/containerinsights/cluster_name/prometheus.

- For the second section, the ECS tasks with `appmesh` in their ECS task definition ARN and with `version` of `:23` are filtered for the container port scan. For containers with a name of `envoy` that expose the container port on `9901`, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value within these ECS tasks expose the container port on 9404 or 9406, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/prometheus`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/prometheus`, and send the scraped metrics to the `envoy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 4

```
"ecs_service_discovery": {
  "sd_frequency": "5m",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "service_name_list_for_tasks": [
    {
      "sd_job_name": "nginx-prometheus",
      "sd_metrics_path": "/metrics",
      "sd_metrics_ports": "9113",
      "sd_service_name_pattern": "^nginx-.*"
    },
    {
      "sd_job_name": "haproxy-prometheus",
      "sd_metrics_path": "/stats/metrics",
      "sd_container_name_pattern": "^haproxy$",
      "sd_metrics_ports": "8404",
      "sd_service_name_pattern": ".*haproxy-service.*"
    }
  ]
}
```

This example enables ECS service name regular expression-based service discovery. The CloudWatch agent will query the ECS services' metadata every five minutes and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

Two service name regular expresion sections are defined:

- For the first section, the ECS tasks that are associated with ECS services that have names matching the regular expression `^nginx-.*` are filtered for the container port scan. If the containers within these ECS tasks expose the container port on 9113, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/metrics`, and the scraped metrics are sent to the `nginx-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.
- or the second section, the ECS tasks that are associated with ECS services that have names matching the regular expression `.*haproxy-service.*` are filtered for the container port scan. For containers with a name of `haproxy` expose the container port on 8404, the mapped host port along with the private IP of the container are used to create the Prometheus exporter targets. The value of `sd_metrics_path` sets `__metrics_path__` to `/stats/metrics`. So the CloudWatch agent will scrape the Prometheus metrics from `private_ip:host_port/stats/metrics`, and the scraped metrics are sent to the `haproxy-prometheus` log stream in CloudWatch Logs in the log group `/aws/ecs/containerinsights/cluster_name/prometheus`.

Example 5

```
"ecs_service_discovery": {
  "sd_frequency": "1m30s",
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
  "docker_label": {
    "sd_port_label": "MY_PROMETHEUS_EXPORTER_PORT_LABEL",
    "sd_metrics_path_label": "MY_PROMETHEUS_METRICS_PATH_LABEL",
    "sd_job_name_label": "MY_PROMETHEUS_METRICS_NAME_LABEL"
  }
  "task_definition_list": [
    {
      "sd_metrics_ports": "9150",
      "sd_task_definition_arn_pattern": "*memcached.*"
    }
  ]
}
```

This example enables both ECS service discovery modes. The CloudWatch agent will query the ECS tasks' metadata every 90 seconds and write the discovered targets into the `/tmp/cwagent_ecs_auto_sd.yaml` file within the CloudWatch agent container.

For the Docker-based service discovery configuration:

- The ECS tasks with Docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` will be filtered for Prometheus port scan. The target Prometheus container port is specified by the value of the label `MY_PROMETHEUS_EXPORTER_PORT_LABEL`.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used for `__metrics_path__`. If the container does not have this docker label, the default value `/metrics` is used.
- The value of the docker label `MY_PROMETHEUS_EXPORTER_PORT_LABEL` is used as the job label. If the container does not have this Docker label, the job name defined in the Prometheus configuration is used.

For the ECS task definition ARN regular expression-based service discovery configuration:

- The ECS tasks with `memcached` in the ECS task definition ARN are filtered for container port scan. The target Prometheus container port is 9150 as defined by `sd_metrics_ports`. The default metrics path `/metrics` is used. The job name defined in the Prometheus configuration is used.

(Optional) Set up sample containerized Amazon ECS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see Prometheus metrics collected by the CloudWatch agent.

Topics

- Sample App Mesh workload for Amazon ECS clusters
- Sample Java/JMX workload for Amazon ECS clusters
- Sample NGINX workload for Amazon ECS clusters
- Sample NGINX Plus workload for Amazon ECS clusters
- Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS

- Tutorial for scraping Redis OSS Prometheus metrics on Amazon ECS Fargate

Sample App Mesh workload for Amazon ECS clusters

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Amazon ECS.

First, follow this walkthrough to deploy the sample color app on your Amazon ECS cluster. After you finish, you will have App Mesh Prometheus metrics exposed on port 9901.

Next, follow these steps to install the CloudWatch agent with Prometheus monitoring on the same Amazon ECS cluster where you installed the color app. The steps in this section install the CloudWatch agent in bridge network mode.

The environment variables `ENVIRONMENT_NAME`, `AWS_PROFILE`, and `AWS_DEFAULT_REGION` that you set in the walkthrough will also be used in the following steps.

To install the CloudWatch agent with Prometheus monitoring for testing

1. Download the CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
bridge-host.yaml
```

2. Set the network mode by entering the following commands.

```
export ECS_CLUSTER_NAME=${ENVIRONMENT_NAME}
export ECS_NETWORK_MODE=bridge
```

3. Create the CloudFormation stack by entering the following commands.

```
aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=${ECS_CLUSTER_NAME} \
                 ParameterKey=CreateIAMRoles,ParameterValue=True \
                 ParameterKey=ECSNetworkMode,ParameterValue=${ECS_NETWORK_MODE} \
                 ParameterKey=TaskRoleName,ParameterValue=CWAgent-Prometheus-
TaskRole-${ECS_CLUSTER_NAME} \
```

```
                  ParameterKey=ExecutionRoleName,ParameterValue=CWAgent-Prometheus-
ExecutionRole-${ECS_CLUSTER_NAME} \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --profile ${AWS_PROFILE}
```

4. (Optional) When the CloudFormation stack is created, you see a `CREATE_COMPLETE` message. If you to check the status before you see that message, enter the following command.

```
aws cloudformation describe-stacks \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--query 'Stacks[0].StackStatus' \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Troubleshooting

The steps in the walkthrough use jq to parse the output result of the AWS CLI. For more information about installing jq, see jq. Use the following command to set the default output format of your AWS CLI to JSON so jq can parse it correctly.

```
$ aws configure
```

When the response gets to `Default output format`, enter **`json`**.

Uninstall the CloudWatch agent with Prometheus monitoring

When you are finished testing, enter the following command to uninstall the CloudWatch agent by deleting the CloudFormation stack.

```
aws cloudformation delete-stack \
--stack-name CWAgent-Prometheus-ECS-${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
--region ${AWS_DEFAULT_REGION} \
--profile ${AWS_PROFILE}
```

Sample Java/JMX workload for Amazon ECS clusters

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

The CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the JMX Exporter to expose the metrics on a different port or metrics_path. If you do change the port or path, update the default `ecs_service_discovery` section in the CloudWatch agent configuration.

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Amazon ECS.

To install the Java/JMX sample workload for Amazon ECS clusters

1. Follow the steps in these sections to create your Docker images.
   - Example: Java Jar Application Docker image with Prometheus metrics
   - Example: Apache Tomcat Docker image with Prometheus metrics
2. Specify the following two Docker labels in the Amazon ECS task definition file. You can then run the task definition as an Amazon ECS service or Amazon ECS task in the cluster.
   - Set `ECS_PROMETHEUS_EXPORTER_PORT` to point to the containerPort where the Prometheus metrics are exposed.
   - Set `Java_EMF_Metrics` to `true`. The CloudWatch agent uses this flag to generated the embedded metric format in the log event.

   The following is an example:

```
{
  "family": "workload-java-ec2-bridge",
  "taskRoleArn": "{{task-role-arn}}",
  "executionRoleArn": "{{execution-role-arn}}",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "tomcat-prometheus-workload-java-ec2-bridge-dynamic-port",
      "image": "your_docker_image_tag_for_tomcat_with_prometheus_metrics",
      "portMappings": [
        {
          "hostPort": 0,
          "protocol": "tcp",
          "containerPort": 9404
```

```
        }
      ],
      "dockerLabels": {
        "ECS_PROMETHEUS_EXPORTER_PORT": "9404",
        "Java_EMF_Metrics": "true"
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"  ],
  "cpu": "256",
  "memory": "512"
  }
```

The default setting of the CloudWatch agent in the CloudFormation template enables both docker label-based service discovery and task definition ARN-based service discovery. To view these default settings, see line 65 of the CloudWatch agent YAML configuration file. The containers with the `ECS_PROMETHEUS_EXPORTER_PORT` label will be auto-discovered based on the specified container port for Prometheus scraping.

The default setting of the CloudWatch agent also has the `metric_declaration` setting for Java/JMX at line 112 of the same file. All Docker labels of the target containers will be added as additional labels in the Prometheus metrics and sent to CloudWatch Logs. For the Java/JMX containers with Docker label `Java_EMF_Metrics="true"`, the embedded metric format will be generated.

Sample NGINX workload for Amazon ECS clusters

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see nginx-prometheus-exporter on Github. For more information about the NGINX reverse proxy, see ecs-nginx-reverse-proxy on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/.
2. Find the app directory and build an image from that directory:

```
docker build -t web-server-app ./path-to-app-directory
```

3. Build a custom image for NGINX. First, create a directory with the following two files:

   - A sample Dockerfile:

```
FROM nginx
COPY nginx.conf /etc/nginx/nginx.conf
```

   - An `nginx.conf` file, modified from https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/:

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  server{
    listen 8080;
    location /stub_status {
        stub_status   on;
    }
  }
```

```
  server {
    listen 80;

    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://app:3000;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_cache_bypass $http_upgrade;
    }
  }
}
```

> **Note**
>
> `stub_status` must be enabled on the same port that `nginx-prometheus-exporter` is configured to scrape metrics from. In our example task definition, `nginx-prometheus-exporter` is configured to scrape metrics from port 8080.

4. Build an image from files in your new directory:

```
docker build -t nginx-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`.

The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-iamge* with the image URI for your customized NGINX image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
        }
      ],
      "links": [
        "app"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 256,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.scrape-uri",
        "http://nginx:8080/stub_status"
```

```
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
  "placementConstraints": [],
  "family": "nginx-sample-stack"
}
```

2. Register the task definition by entering the following command.

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. Create a service to run the task by entering the following command:

   Be sure not to change the service name. You will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-service$`. The next section provides more details.

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-service \
 --task-definition nginx-sample-stack:1 \
 --desired-count 1
```

Configure the CloudWatch agent to scrape NGINX Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task through the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the

metrics available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent confguration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-service$"
   }
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["job"],
  "label_matcher": ".*nginx.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginx_.*$"
  ]
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

   If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-scraping-support
```

5. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.

6. Revew the newly-created changeset **nginx-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restart the CloudWatch agent task by entering the following command:

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name
```

```
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your NGINX metrics and logs

You can now view the NGINX metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.
3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/*your_cluster_name*/prometheus**, in the log stream *nginx-prometheus-exporter*.

Sample NGINX Plus workload for Amazon ECS clusters

NGINX Plus is the commerical version of NGINX. You must have a licence to use it. For more information, see NGINX Plus

The NGINX Prometheus exporter can scrape and expose NGINX data as Prometheus metrics. This example uses the exporter in tandem with the NGINX Plus reverse proxy service for Amazon ECS.

For more information about the NGINX Prometheus exporter, see nginx-prometheus-exporter on Github. For more information about the NGINX reverse proxy, see ecs-nginx-reverse-proxy on Github.

The CloudWatch agent with Prometheus support scrapes the NGINX Plus Prometheus metrics based on the service discovery configuration in the Amazon ECS cluster. You can configure the NGINX Prometheus Exporter to expose the metrics on a different port or path. If you change the port or path, update the `ecs_service_discovery` section in the CloudWatch agent configuration file.

Install the NGINX Plus reverse proxy sample workload for Amazon ECS clusters

Follow these steps to install the NGINX reverse proxy sample workload.

Create the Docker images

To create the Docker images for the NGINX Plus reverse proxy sample workload

1. Download the following folder from the NGINX reverse proxy repo: https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/.
2. Find the app directory and build an image from that directory:

   ```
   docker build -t web-server-app ./path-to-app-directory
   ```

3. Build a custom image for NGINX Plus. Before you can build the image for NGINX Plus, you need to obtain the key named `nginx-repo.key` and the SSL certificate `nginx-repo.crt` for your licensed NGINX Plus. Create a directory and store in it your `nginx-repo.key` and `nginx-repo.crt` files.

   In the directory that you just created, create the following two files:

   - A sample Dockerfile with the following content. This Docker file is adopted from a sample file provided at https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/#docker_plus_image. The important change that we make is that we load a separate file, called `nginx.conf`, which will be created in the next step.

     ```
     FROM debian:buster-slim

     LABEL maintainer="NGINX Docker Maintainers <docker-maint@nginx.com>"

     # Define NGINX versions for NGINX Plus and NGINX Plus modules
     ```

```
# Uncomment this block and the versioned nginxPackages block in the main RUN
# instruction to install a specific release
# ENV NGINX_VERSION 21
# ENV NJS_VERSION 0.3.9
# ENV PKG_RELEASE 1~buster

# Download certificate and key from the customer portal (https://cs.nginx.com
 (https://cs.nginx.com/))
# and copy to the build context
COPY nginx-repo.crt /etc/ssl/nginx/
COPY nginx-repo.key /etc/ssl/nginx/
# COPY nginx.conf /etc/ssl/nginx/nginx.conf

RUN set -x \
# Create nginx user/group first, to be consistent throughout Docker variants
&& addgroup --system --gid 101 nginx \
&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /
nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y ca-
certificates gnupg1 \
&& \
NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
ha.pool.sks-keyservers.net (http://ha.pool.sks-keyservers.net/) \
hkp://keyserver.ubuntu.com:80 \
hkp://p80.pool.sks-keyservers.net:80 \
pgp.mit.edu (http://pgp.mit.edu/) \
; do \
echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-keys
 "$NGINX_GPGKEY" && found=yes && break; \
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" &&
 exit 1; \
apt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \
# Install the latest release of NGINX Plus and/or NGINX Plus modules
# Uncomment individual modules if necessary
# Use versioned packages over defaults to specify a release
&& nginxPackages=" \
nginx-plus \
# nginx-plus=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-xslt \
```

```
# nginx-plus-module-xslt=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-geoip \
# nginx-plus-module-geoip=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-image-filter \
# nginx-plus-module-image-filter=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-perl \
# nginx-plus-module-perl=${NGINX_VERSION}-${PKG_RELEASE} \
# nginx-plus-module-njs \
# nginx-plus-module-njs=${NGINX_VERSION}+${NJS_VERSION}-${PKG_RELEASE} \
" \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Peer \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::Verify-Host \"true\";" >> /etc/apt/
apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslCert \"/etc/ssl/nginx/nginx-
repo.crt\";" >> /etc/apt/apt.conf.d/90nginx \
&& echo "Acquire::https::plus-pkgs.nginx.com::SslKey \"/etc/ssl/nginx/nginx-
repo.key\";" >> /etc/apt/apt.conf.d/90nginx \
&& printf "deb https://plus-pkgs.nginx.com/debian buster nginx-plus\n" > /etc/
apt/sources.list.d/nginx-plus.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
$nginxPackages \
gettext-base \
curl \
&& apt-get remove --purge --auto-remove -y && rm -rf /var/lib/apt/lists/* /etc/
apt/sources.list.d/nginx-plus.list \
&& rm -rf /etc/apt/apt.conf.d/90nginx /etc/ssl/nginx

# Forward request logs to Docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log

COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
```

- An `nginx.conf` file, modified from https://github.com/awslabs/ecs-nginx-reverse-proxy/tree/master/reverse-proxy/nginx.

```
events {
  worker_connections 768;
}

http {
  # Nginx will handle gzip compression of responses from the app server
  gzip on;
  gzip_proxied any;
  gzip_types text/plain application/json;
  gzip_min_length 1000;

  upstream backend {
    zone name 10m;
    server app:3000    weight=2;
    server app2:3000    weight=1;
  }

  server{
    listen 8080;
    location /api {
      api write=on;
    }
  }

  match server_ok {
    status 100-599;
  }

  server {
    listen 80;
    status_zone zone;
    # Nginx will reject anything not matching /api
    location /api {
      # Reject requests with unsupported HTTP method
      if ($request_method !~ ^(GET|POST|HEAD|OPTIONS|PUT|DELETE)$) {
        return 405;
      }

      # Only requests matching the whitelist expectations will
      # get sent to the application server
      proxy_pass http://backend;
      health_check uri=/lorem-ipsum match=server_ok;
      proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_cache_bypass $http_upgrade;
      }
    }
}
```

4. Build an image from files in your new directory:

```
docker build -t nginx-plus-reverse-proxy ./path-to-your-directory
```

5. Upload your new images to an image repository for later use.

Create the task definition to run NGINX Plus and the web server app in Amazon ECS

Next, you set up the task definition.

This task definition enables the collection and export of NGINX Plus Prometheus metrics. The NGINX container tracks input from the app, and exposes that data to port 8080, as set in `nginx.conf`. The NGINX prometheus exporter container scrapes these metrics, and posts them to port 9113, for use in CloudWatch.

To set up the task definition for the NGINX sample Amazon ECS workload

1. Create a task definition JSON file with the following content. Replace *your-customized-nginx-plus-image* with the image URI for your customized NGINX Plus image, and replace *your-web-server-app-image* with the image URI for your web server app image.

```
{
  "containerDefinitions": [
    {
      "name": "nginx",
      "image": "your-customized-nginx-plus-image",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp"
```

```
        }
      ],
      "links": [
        "app",
        "app2"
      ]
    },
    {
      "name": "app",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "app2",
      "image": "your-web-server-app-image",
      "memory": 256,
      "cpu": 128,
      "essential": true
    },
    {
      "name": "nginx-prometheus-exporter",
      "image": "docker.io/nginx/nginx-prometheus-exporter:0.8.0",
      "memory": 256,
      "cpu": 256,
      "essential": true,
      "command": [
        "-nginx.plus",
        "-nginx.scrape-uri",
         "http://nginx:8080/api"
    ],
    "links":[
      "nginx"
    ],
      "portMappings":[
        {
          "containerPort": 9113,
          "protocol": "tcp"
        }
      ]
    }
  ],
  "networkMode": "bridge",
```

```
  "placementConstraints": [],
  "family": "nginx-plus-sample-stack"
}
```

2. Register the task definition:

```
aws ecs register-task-definition --cli-input-json file://path-to-your-task-definition-json
```

3. Create a service to run the task by entering the following command:

```
aws ecs create-service \
 --cluster your-cluster-name \
 --service-name nginx-plus-service \
 --task-definition nginx-plus-sample-stack:1 \
 --desired-count 1
```

Be sure not to change the service name. We will be running a CloudWatch agent service using a configuration that searches for tasks using the name patterns of the services that started them. For example, for the CloudWatch agent to find the task launched by this command, you can specify the value of `sd_service_name_pattern` to be `^nginx-plus-service$`. The next section provides more details.

Configure the CloudWatch agent to scrape NGINX Plus Prometheus metrics

The final step is to configure the CloudWatch agent to scrape the NGINX metrics. In this example, the CloudWatch agent discovers the task through the service name pattern, and the port 9113, where the exporter exposes the prometheus metrics for NGINX. With the task discovered and the metrics available, the CloudWatch agent begins posting the collected metrics to the log stream **nginx-prometheus-exporter**.

To configure the CloudWatch agent to scrape the NGINX metrics

1. Download the latest version of the necessary YAML file by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-bridge-host.yaml
```

2. Open the file with a text editor, and find the full CloudWatch agent confguration in the `value` key in the `resource:CWAgentConfigSSMParameter` section. Then, in the `ecs_service_discovery` section, add the following `service_name_list_for_tasks` section.

```
"service_name_list_for_tasks": [
  {
    "sd_job_name": "nginx-plus-prometheus-exporter",
    "sd_metrics_path": "/metrics",
    "sd_metrics_ports": "9113",
    "sd_service_name_pattern": "^nginx-plus.*"
   }
],
```

3. In the same file, add the following section in the `metric_declaration` section to allow NGINX Plus metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName"]],
  "metric_selectors": [
    "^nginxplus_connections_accepted$",
    "^nginxplus_connections_active$",
    "^nginxplus_connections_dropped$",
    "^nginxplus_connections_idle$",
    "^nginxplus_http_requests_total$",
    "^nginxplus_ssl_handshakes$",
    "^nginxplus_ssl_handshakes_failed$",
    "^nginxplus_up$",
    "^nginxplus_upstream_server_health_checks_fails$"
  ]
},
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName",
 "upstream"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_response_time$"
  ]
},
```

```
{
  "source_labels": ["job"],
  "label_matcher": "^nginx-plus.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily", "ServiceName", "code"]],
  "metric_selectors": [
    "^nginxplus_upstream_server_responses$",
    "^nginxplus_server_zone_responses$"
  ]
},
```

4. If you don't already have the CloudWatch agent deployed in this cluster, skip to step 8.

   If you already have the CloudWatch agent deployed in the Amazon ECS cluster by using AWS CloudFormation, you can create a change set by entering the following commands:

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name nginx-plus-scraping-support
```

5. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.

6. Revew the newly-created changeset **nginx-plus-scraping-support**. You should see one change applied to the **CWAgentConfigSSMParameter** resource. Run the changeset and restrt the CloudWatch agent task by entering the following command:

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
```

```
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

8. If you are installing the CloudWatch agent with Prometheus metric collecting on the cluster for the first time, enter the following commands.

```
ECS_CLUSTER_NAME=your_cluster_name
AWS_REGION=your_aws_region
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your NGINX Plus metrics and logs

You can now view the NGINX Plus metrics being collected.

To view the metrics for your sample NGINX workload

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the Region where your cluster is running, choose **Metrics** in the left navigation pane. Find the **ContainerInsights/Prometheus** namespace to see the metrics.

3. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group **/aws/containerinsights/*your_cluster_name*/prometheus**, in the log stream *nginx-plus-prometheus-exporter*.

Tutorial for adding a new Prometheus scrape target: Memcached on Amazon ECS

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Memcached application on an Amazon Amazon ECS cluster with the EC2 launch type. The Memcached Prometheus exporter target will be auto-discovered by the CloudWatch agent by ECS task definition-based service discovery.

Memcached is a general-purpose distributed memory caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. For more infromation, see What is Memcached?

The memchached_exporter (Apache License 2.0) is one of the Prometheus official exporters. By default the memcache_exporter serves on port 0.0.0.0:9150 at `/metrics`.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- Memcached
- prom/memcached-exporter

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Amazon ECS.

Topics

- Set the Amazon ECS EC2 cluster environment variables
- Install the sample Memcached workload
- Configure the CloudWatch agent to scrape Memcached Prometheus metrics
- Viewing your Memcached metrics

Set the Amazon ECS EC2 cluster environment variables

To set the Amazon ECS EC2 cluster environment variables

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see Installing the Amazon ECS CLI.
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-ec2-memcached-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS cluster with the EC2 launch type where you want to install the sample Memcached workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam --size 1 \
--instance-type t3.medium \
--cluster $ECS_CLUSTER_NAME \
--region $AWS_REGION
```

The expected result of this command is as follows:

```
WARN[0000] You will not be able to SSH into your EC2 instances without a key pair.
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.44.4 and Docker
 version 19.03.6-ce
INFO[0001] Created cluster                               cluster=ecs-ec2-memcached-
tutorial region=ca-central-1
INFO[0002] Waiting for your cluster resources to be created...
INFO[0002] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0063] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
INFO[0124] Cloudformation stack status
 stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxxx
Security Group created: sg-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Install the sample Memcached workload

To install the sample Memcached workload which exposes the Prometheus metrics

1. Download the Memcached CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/memcached/memcached-traffic-sample.yaml
```

2. Set the IAM role names to be created for Memcached by entering the following commands.

```
MEMCACHED_ECS_TASK_ROLE_NAME=memcached-prometheus-demo-ecs-task-role-name
MEMCACHED_ECS_EXECUTION_ROLE_NAME=memcached-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Memcached workload by entering the following command. This sample installs the workload in `host` network mode.

```
MEMCACHED_ECS_NETWORK_MODE=host

aws cloudformation create-stack --stack-name Memcached-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-EC2-$MEMCACHED_ECS_NETWORK_MODE \
    --template-body file://memcached-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=ECSNetworkMode,ParameterValue=
$MEMCACHED_ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=
$MEMCACHED_ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$MEMCACHED_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

The CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Memcached task definition
- One Memcached service

In the Memcached task definition, two containers are defined:

- The primary container runs a simple Memcached application and opens port 11211 for access.
- The other container runs the Redis OSS exporter process to expose the Prometheus metrics on port 9150. This is the container to be discovered and scraped by the CloudWatch agent.

Configure the CloudWatch agent to scrape Memcached Prometheus metrics

To configure the CloudWatch agent to scrape Memcached Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-awsvpc.yaml
   ```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

   Then, in the `ecs_service_discovery` section, add the following configuration into the `task_definition_list` section.

   ```
   {
       "sd_job_name": "ecs-memcached",
       "sd_metrics_ports": "9150",
       "sd_task_definition_arn_pattern": ".*:task-definition/memcached-prometheus-demo.*:[0-9]+"
   },
   ```

   For the `metric_declaration` section, the default setting does not allow any Memcached metrics. Add the following section to allow Memcached metrics. Be sure to follow the existing indentation pattern.

   ```
   {
     "source_labels": ["container_name"],
     "label_matcher": "memcached-exporter-.*",
     "dimensions": [["ClusterName", "TaskDefinitionFamily"]],
     "metric_selectors": [
   ```

```
    "^memcached_current_(bytes|items|connections)$",
    "^memcached_items_(reclaimed|evicted)_total$",
    "^memcached_(written|read)_bytes_total$",
    "^memcached_limit_bytes$",
    "^memcached_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "memcached-exporter-.*",
  "dimensions": [["ClusterName", "TaskDefinitionFamily","status","command"],
 ["ClusterName", "TaskDefinitionFamily","command"]],
  "metric_selectors": [
    "^memcached_commands_total$"
  ]
},
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by CloudFormation, you can create a change set by entering the following commands.

```
ECS_NETWORK_MODE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name memcached-scraping-support
```

4. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.

5. Review the newly created changeset `memcached-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-EC2-$ECS_NETWORK_MODE \
--region $AWS_REGION
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_NETWORK_MODEE=bridge
CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION
```

Viewing your Memcached metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions
memcached_current_items	ClusterName ,TaskDefinitionFamily
memcached_current_connections	ClusterName ,TaskDefinitionFamily
memcached_limit_bytes	ClusterName ,TaskDefinitionFamily
memcached_current_bytes	ClusterName ,TaskDefinitionFamily
memcached_written_bytes_total	ClusterName ,TaskDefinitionFamily
memcached_read_bytes_total	ClusterName ,TaskDefinitionFamily
memcached_items_evicted_total	ClusterName ,TaskDefinitionFamily
memcached_items_reclaimed_total	ClusterName ,TaskDefinitionFamily
memcached_commands_total	ClusterName ,TaskDefinitionFamily ClusterName ,TaskDefinitionFamily, command ClusterName ,TaskDefinitionFamily, status, command

> **Note**
>
> The value of the **command** dimension can be: `delete`, `get`, `cas`, `set`, `decr`, `touch`, `incr`, or `flush`.
> The value of the **status** dimension can be `hit`, `miss`, or `badval`.

You can also create a CloudWatch dashboard for your Memcached Prometheus metrics.

To create a dashboard for Memcached Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_memcached_cw_dashboard_name
ECS_TASK_DEF_FAMILY=memcached-prometheus-demo-$ECS_CLUSTER_NAME-EC2-
$MEMCACHED_ECS_NETWORK_MOD
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
container-insights/latest/ecs-task-definition-templates/deployment-mode/
replica-service/cwagent-prometheus/sample_cloudwatch_dashboards/memcached/
cw_dashboard_memcached.json \
| sed "s/{{YOUR_AWS_REGION}}/$AWS_REGION/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/$ECS_CLUSTER_NAME/g" \
| sed "s/{{YOUR_TASK_DEF_FAMILY}}/$ECS_TASK_DEF_FAMILY/g" \
| xargs -0 aws cloudwatch put-dashboard --dashboard-name ${DASHBOARD_NAME} --region
 $AWS_REGION --dashboard-body
```

Tutorial for scraping Redis OSS Prometheus metrics on Amazon ECS Fargate

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis OSS application in an Amazon ECS Fargate cluster. The Redis OSS Prometheus exporter target will be auto-discovered by the CloudWatch agent with Prometheus metric support based on the container's Docker labels.

Redis OSS (https://redis.io/) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see redis.

redis_exporter (MIT License licensed) is used to expose the Redis OSS prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see redis_exporter.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- redis
- redis_exporter

Prerequisite

To collect metrics from a sample Prometheus workload for Amazon ECS, you must be running Container Insights in the cluster. For information about installing Container Insights, see Amazon ECS.

Topics

- Set the Amazon ECS Fargate cluster environment variable
- Set the network environment variables for the Amazon ECS Fargate cluster
- Install the sample Redis OSS workload
- Configure the CloudWatch agent to scrape Redis OSS Prometheus metrics
- Viewing your Redis OSS metrics

Set the Amazon ECS Fargate cluster environment variable

To set the Amazon ECS Fargate cluster environment variable

1. Install the Amazon ECS CLI if you haven't already done so. For more information, see Installing the Amazon ECS CLI.
2. Set the new Amazon ECS cluster name and Region. For example:

```
ECS_CLUSTER_NAME=ecs-fargate-redis-tutorial
AWS_DEFAULT_REGION=ca-central-1
```

3. (Optional) If you don't already have an Amazon ECS Fargate cluster where you want to install the sample Redis OSS workload and CloudWatch agent, you can create one by entering the following command.

```
ecs-cli up --capability-iam \
```

```
--cluster $ECS_CLUSTER_NAME \
--launch-type FARGATE \
--region $AWS_DEFAULT_REGION
```

The expected result of this command is as follows:

```
INFO[0000] Created cluster   cluster=ecs-fargate-redis-tutorial region=ca-central-1
INFO[0001] Waiting for your cluster resources to be created...
INFO[0001] Cloudformation stack status   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-xxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Subnet created: subnet-xxxxxxxxxxxxxxxx
Cluster creation succeeded.
```

Set the network environment variables for the Amazon ECS Fargate cluster

To set the network environment variables for the Amazon ECS Fargate cluster

1. Set your VPC and subnet ID of the Amazon ECS cluster. If you created a new cluster in the previous procedure, you'll see these values in the result of the final command. Otherwise, use the IDs of the existing cluster that you are going to use with Redis.

   ```
   ECS_CLUSTER_VPC=vpc-xxxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_1=subnet-xxxxxxxxxxxxxxxxx
   ECS_CLUSTER_SUBNET_2=subnet-xxxxxxxxxxxxxxxxx
   ```

2. In this tutorial, you install the Redis OSS application and the CloudWatch agent in the default security group of the Amazon ECS cluster's VPC. The default security group allows all network connection within the same security group so the CloudWatch agent can scrape the Prometheus metrics exposed on the Redis OSS containers. In a real production environment, you might want to create dedicated security groups for the Redis OSS application and CloudWatch agent and set customized permissions for them.

   Enter the following command to get the default security group ID.

   ```
   aws ec2 describe-security-groups \
   --filters Name=vpc-id,Values=$ECS_CLUSTER_VPC  \
   --region $AWS_DEFAULT_REGION
   ```

Then set the Fargate cluster deafult security group variable by entering the following command, replacing *my-default-security-group* with the value you found from the previous command.

```
ECS_CLUSTER_SECURITY_GROUP=my-default-security-group
```

Install the sample Redis OSS workload

To install the sample Redis OSS workload which exposes the Prometheus metrics

1. Download the Redis OSS CloudFormation template by entering the following command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
cwagent-prometheus/sample_traffic/redis/redis-traffic-sample.yaml
```

2. Set the IAM role names to be created for Redis OSS by entering the following commands.

```
REDIS_ECS_TASK_ROLE_NAME=redis-prometheus-demo-ecs-task-role-name
REDIS_ECS_EXECUTION_ROLE_NAME=redis-prometheus-demo-ecs-execution-role-name
```

3. Install the sample Redis OSS workload by entering the following command.

```
aws cloudformation create-stack --stack-name Redis-Prometheus-Demo-ECS-
$ECS_CLUSTER_NAME-fargate-awsvpc \
    --template-body file://redis-traffic-sample.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET_1 \
                 ParameterKey=TaskRoleName,ParameterValue=$REDIS_ECS_TASK_ROLE_NAME
 \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$REDIS_ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_DEFAULT_REGION
```

The CloudFormation stack creates four resources:

- One ECS task role
- One ECS task execution role
- One Redis OSS task definition
- One Redis OSS service

In the Redis OSS task definition, two containers are defined:

- The primary container runs a simple Redis OSS application and opens port 6379 for access.
- The other container runs the Redis OSS exporter process to expose the Prometheus metrics on port 9121. This is the container to be discovered and scraped by the CloudWatch agent. The following Docker label is defined so that the CloudWatch agent can discover this container based on it.

```
ECS_PROMETHEUS_EXPORTER_PORT: 9121
```

Configure the CloudWatch agent to scrape Redis OSS Prometheus metrics

To configure the CloudWatch agent to scrape Redis OSS Prometheus metrics

1. Download the latest version of `cwagent-ecs-prometheus-metric-for-awsvpc.yaml` by entering the following command.

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
   insights/latest/ecs-task-definition-templates/deployment-mode/replica-service/
   cwagent-prometheus/cloudformation-quickstart/cwagent-ecs-prometheus-metric-for-
   awsvpc.yaml
   ```

2. Open the file with a text editor, and find the full CloudWatch agent configuration behind the `value` key in the `resource:CWAgentConfigSSMParameter` section.

   Then, in the `ecs_service_discovery` section shown here, the `docker_label`-based service discovery is enabled with the default settings which are based on `ECS_PROMETHEUS_EXPORTER_PORT`, which matches the Docker label we defined in the Redis OSS ECS task definition. So we do not need to make any changes in this section:

   ```
   ecs_service_discovery": {
     "sd_frequency": "1m",
   ```

```
  "sd_result_file": "/tmp/cwagent_ecs_auto_sd.yaml",
*  "docker_label": {
  },*
  ...
```

For the `metric_declaration` section, the default setting does not allow any Redis OSS metrics. Add the following section to allow Redis OSS metrics. Be sure to follow the existing indentation pattern.

```
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["container_name"],
  "label_matcher": "^redis-exporter-.*$",
  "dimensions": [["ClusterName","TaskDefinitionFamily","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

3. If you already have the CloudWatch agent deployed in the Amazon ECS cluster by CloudFormation, you can create a change set by entering the following commands.

```
ECS_LAUNCH_TYPE=FARGATE
```

```
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION} \
    --change-set-name redis-scraping-support
```

4. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.

5. Review the newly created changeset `redis-scraping-support`. You should see one change applied to the `CWAgentConfigSSMParameter` resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 0 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

6. Wait about 10 seconds, and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service cwagent-prometheus-replica-service-$ECS_LAUNCH_TYPE-awsvpc \
--region ${AWS_DEFAULT_REGION}
```

7. If you are installing the CloudWatch agent with Prometheus metric collecting for the cluster for the first time, please enter the following commands:

```
ECS_LAUNCH_TYPE=FARGATE
```

```
CREATE_IAM_ROLES=True
ECS_CLUSTER_SUBNET=$ECS_CLUSTER_SUBNET_1
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-stack --stack-name CWAgent-Prometheus-ECS-
$ECS_CLUSTER_NAME-$ECS_LAUNCH_TYPE-awsvpc \
    --template-body file://cwagent-ecs-prometheus-metric-for-awsvpc.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSLaunchType,ParameterValue=$ECS_LAUNCH_TYPE \
                 ParameterKey=SecurityGroupID,ParameterValue=
$ECS_CLUSTER_SECURITY_GROUP \
                 ParameterKey=SubnetID,ParameterValue=$ECS_CLUSTER_SUBNET \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=
$ECS_EXECUTION_ROLE_NAME \
    --capabilities CAPABILITY_NAMED_IAM \
    --region ${AWS_DEFAULT_REGION}
```

Viewing your Redis OSS metrics

This tutorial sends the following metrics to the **ECS/ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric Name	Dimensions	
redis_net_input_bytes_total	ClusterName, TaskDefinitionFamily	
redis_net_output_bytes_total	ClusterName, TaskDefinitionFamily	
redis_expired_keys_total	ClusterName, TaskDefinitionFamily	

Metric Name	Dimensions	
redis_evicted_keys_total	ClusterName, TaskDefinitionFamily	
redis_keyspace_hits_total	ClusterName, TaskDefinitionFamily	
redis_keyspace_misses_total	ClusterName, TaskDefinitionFamily	
redis_memory_used_bytes	ClusterName, TaskDefinitionFamily	
redis_connected_clients	ClusterName, TaskDefinitionFamily	
redis_commands_total	ClusterName ,TaskDefinitionFamily , cmd	
redis_db_keys	ClusterName ,TaskDefinitionFamily , db	

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.
The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis OSS Prometheus metrics.

To create a dashboard for Redis OSS Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
ECS_TASK_DEF_FAMILY=redis-prometheus-demo-$ECS_CLUSTER_NAME-fargate-awsvpc
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Set up and configure Prometheus metrics collection on Amazon EKS and Kubernetes clusters

To collect Prometheus metrics from clusters running Amazon EKS or Kubernetes, you can use the CloudWatch agent as a collector or use the AWS Distro for OpenTelemetry collector. For information about using the AWS Distro for OpenTelemetry collector, see https://aws-otel.github.io/docs/getting-started/container-insights/eks-prometheus.

The following sections explain how to collect Prometheus metrics using the CloudWatch agent. They explain how to install the CloudWatch agent with Prometheus monitoring on clusters running Amazon EKS or Kubernetes, and how to configure the agent to scrape additional targets. They also provide optional tutorials for setting up sample workloads to use for testing with Prometheus monitoring.

Topics

- Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX

- Memcached
- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters
- Scraping additional Prometheus sources and importing those metrics
- (Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing

Install the CloudWatch agent with Prometheus metrics collection on Amazon EKS and Kubernetes clusters

This section explains how to set up the CloudWatch agent with Prometheus monitoring in a cluster running Amazon EKS or Kubernetes. After you do this, the agent automatically scrapes and imports metrics for the following workloads running in that cluster.

- AWS App Mesh
- NGINX
- Memcached

- Java/JMX
- HAProxy
- Fluent Bit

You can also configure the agent to scrape and import additional Prometheus workloads and sources.

Before following these steps to install the CloudWatch agent for Prometheus metric collection, you must have a cluster running on Amazon EKS or a Kubernetes cluster running on an Amazon EC2 instance.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Topics

- Setting up IAM roles
- Installing the CloudWatch agent to collect Prometheus metrics

Setting up IAM roles

The first step is to set up the necessary IAM role in the cluster. There are two methods:

- Set up an IAM role for a service account, also known as a *service role*. This method works for both the EC2 launch type and the Fargate launch type.
- Add an IAM policy to the IAM role used for the cluster. This works only for the EC2 launch type.

Set up a service role (EC2 launch type and Fargate launch type)

To set up a service role, enter the following command. Replace *MyCluster* with the name of the cluster.

```
eksctl create iamserviceaccount \
 --name cwagent-prometheus \
```

```
--namespace amazon-cloudwatch \
 --cluster MyCluster \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

Add a policy to the node group's IAM role (EC2 launch type only)

To set up the IAM policy in a node group for Prometheus support

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. You need to find the prefix of the IAM role name for the cluster. To do this, select the check box next to the name of an instance that is in the cluster, and choose **Actions**, **Security**, **Modify IAM Role**. Then copy the prefix of the IAM role, such as `eksctl-dev303-workshop-nodegroup`.
4. Open the IAM console at https://console.aws.amazon.com/iam/.
5. In the navigation pane, choose **Roles**.
6. Use the search box to find the prefix that you copied earlier in this procedure, and choose that role.
7. Choose **Attach policies**.
8. Use the search box to find **CloudWatchAgentServerPolicy**. Select the check box next to **CloudWatchAgentServerPolicy**, and choose **Attach policy**.

Installing the CloudWatch agent to collect Prometheus metrics

You must install the CloudWatch agent in the cluster to collect the metrics. How to install the agent differs for Amazon EKS clusters and Kubernetes clusters.

Delete previous versions of the CloudWatch agent with Prometheus support

If you have already installed a version of the CloudWatch agent with Prometheus support in your cluster, you must delete that version by entering the following command. This is necessary only for previous versions of the agent with Prometheus support. You do not need to delete the CloudWatch agent that enables Container Insights without Prometheus support.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

Installing the CloudWatch agent on Amazon EKS clusters with the EC2 launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster

1. Enter the following command to check whether the `amazon-cloudwatch` namespace has already been created:

   ```
   kubectl get namespace
   ```

2. If `amazon-cloudwatch` is not displayed in the results, create it by entering the following command:

   ```
   kubectl create namespace amazon-cloudwatch
   ```

3. To deploy the agent with the default configuration and have it send data to the AWS Region that it is installed in, enter the following command:

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
   container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
   service/cwagent-prometheus/prometheus-eks.yaml
   ```

   To have the agent send data to a different Region instead, follow these steps:

   a. Download the YAML file for the agent by entering the following command:

      ```
      curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-
      container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/
      service/cwagent-prometheus/prometheus-eks.yaml
      ```

   b. Open the file with a text editor, and search for the `cwagentconfig.json` block of the file.

   c. Add the highlighted lines, specifying the Region that you want:

      ```
      cwagentconfig.json: |
          {
            "agent": {
              "region": "us-east-2"
            },
      ```

```
      "logs": { ...
```

d. Save the file and deploy the agent using your updated file.

```
kubectl apply -f prometheus-eks.yaml
```

Installing the CloudWatch agent on Amazon EKS clusters with the Fargate launch type

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type, follow these steps.

To install the CloudWatch agent with Prometheus support on an Amazon EKS cluster with the Fargate launch type

1. Enter the following command to create a Fargate profile for the CloudWatch agent so that it can run inside the cluster. Replace *MyCluster* with the name of the cluster.

   ```
   eksctl create fargateprofile --cluster MyCluster \
   --name amazon-cloudwatch \
   --namespace amazon-cloudwatch
   ```

2. To install the CloudWatch agent, enter the following command. Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

   Replace *region* with the name of the Region where you want the metrics to be sent. For example, `us-west-1`.

   ```
   curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
   insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
   prometheus/prometheus-eks-fargate.yaml |
   sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
   kubectl apply -f -
   ```

Installing the CloudWatch agent on a Kubernetes cluster

To install the CloudWatch agent with Prometheus support on a cluster running Kubernetes, enter the following command:

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-k8s.yaml |
sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" |
kubectl apply -f -
```

Replace *MyCluster* with the name of the cluster. This name is used in the log group name that stores the log events collected by the agent, and is also used as a dimension for the metrics collected by the agent.

Replace *region* with the name of the AWS Region where you want the metrics to be sent. For example, **us-west-1**.

Verify that the agent is running

On both Amazon EKS and Kubernetes clusters, you can enter the following command to confirm that the agent is running.

```
kubectl get pod -l "app=cwagent-prometheus" -n amazon-cloudwatch
```

If the results include a single CloudWatch agent pod in the `Running` state, the agent is running and collecting Prometheus metrics. By default the CloudWatch agent collects metrics for App Mesh, NGINX, Memcached, Java/JMX, and HAProxy every minute. For more information about those metrics, see Prometheus metrics collected by the CloudWatch agent. For instructions on how to see your Prometheus metrics in CloudWatch, see Viewing your Prometheus metrics

You can also configure the CloudWatch agent to collect metrics from other Prometheus exporters. For more information, see Scraping additional Prometheus sources and importing those metrics.

Scraping additional Prometheus sources and importing those metrics

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. One is for the standard Prometheus configurations as documented in <scrape_config> in the Prometheus documentation. The other is for the CloudWatch agent configuration.

For Amazon EKS clusters, the configurations are defined in `prometheus-eks.yaml` (for the EC2 launch type) or `prometheus-eks-fargate.yaml` (for the Fargate launch type) as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent. You can use this section to configure how the Prometheus metrics are collected by CloudWatch. For example, you specify which metrics are to be imported into CloudWatch, and define their dimensions.

For Kubernetes clusters running on Amazon EC2 instances, the configurations are defined in the `prometheus-k8s.yaml` YAML file as two config maps:

- The `name: prometheus-config` section contains the settings for Prometheus scraping.
- The `name: prometheus-cwagentconfig` section contains the configuration for the CloudWatch agent.

To scrape additional Prometheus metrics sources and import those metrics to CloudWatch, you modify both the Prometheus scrape configuration and the CloudWatch agent configuration, and then re-deploy the agent with the updated configuration.

VPC security group requirements

The ingress rules of the security groups for the Prometheus workloads must open the Prometheus ports to the CloudWatch agent for scraping the Prometheus metrics by the private IP.

The egress rules of the security group for the CloudWatch agent must allow the CloudWatch agent to connect to the Prometheus workloads' port by private IP.

Prometheus scrape configuration

The CloudWatch agent supports the standard Prometheus scrape configurations as documented in <scrape_config> in the Prometheus documentation. You can edit this section to update the configurations that are already in this file, and add additional Prometheus scraping targets. By default, the sample configuration file contains the following global configuration lines:

```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
```

- **scrape_interval**— Defines how frequently to scrape targets.
- **scrape_timeout**— Defines how long to wait before a scrape request times out.

You can also define different values for these settings at the job level, to override the global configurations.

Prometheus scraping jobs

The CloudWatch agent YAML files already have some default scraping jobs configured. For example, in `prometheus-eks.yaml`, the default scraping jobs are configured in the `job_name` lines in the `scrape_configs` section. In this file, the following default `kubernetes-pod-jmx` section scrapes JMX exporter metrics.

```
- job_name: 'kubernetes-pod-jmx'
   sample_limit: 10000
   metrics_path: /metrics
   kubernetes_sd_configs:
   - role: pod
   relabel_configs:
   - source_labels: [__address__]
     action: keep
     regex: '.*:9404$'
   - action: labelmap
     regex: __meta_kubernetes_pod_label_(.+)
   - action: replace
     source_labels:
     - __meta_kubernetes_namespace
     target_label: Namespace
   - source_labels: [__meta_kubernetes_pod_name]
     action: replace
     target_label: pod_name
   - action: replace
     source_labels:
     - __meta_kubernetes_pod_container_name
     target_label: container_name
   - action: replace
     source_labels:
     - __meta_kubernetes_pod_controller_name
     target_label: pod_controller_name
   - action: replace
     source_labels:
     - __meta_kubernetes_pod_controller_kind
     target_label: pod_controller_kind
   - action: replace
     source_labels:
     - __meta_kubernetes_pod_phase
```

```
        target_label: pod_phase
```

Each of these default targets are scraped, and the metrics are sent to CloudWatch in log events using embedded metric format. For more information, see Embedding metrics within logs.

Log events from Amazon EKS and Kubernetes clusters are stored in the **/aws/containerinsights/*cluster_name*/prometheus** log group in CloudWatch Logs. Log events from Amazon ECS clusters are stored in the **/aws/ecs/containerinsights/*cluster_name*/prometheus** log group.

Each scraping job is contained in a different log stream in this log group. For example, the Prometheus scraping job `kubernetes-pod-appmesh-envoy` is defined for App Mesh. All App Mesh Prometheus metrics from Amazon EKS and Kubernetes clusters are sent to the log stream named **/aws/containerinsights/*cluster_name*>prometheus/kubernetes-pod-appmesh-envoy/**.

To add a new scraping target, you add a new `job_name` section to the `scrape_configs` section of the YAML file, and restart the agent. For an example of this process, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

CloudWatch agent configuration for Prometheus

The CloudWatch agent configuration file has a `prometheus` section under `metrics_collected` for the Prometheus scraping configuration. It includes the following configuration options:

- **cluster_name**— specifies the cluster name to be added as a label in the log event. This field is optional. If you omit it, the agent can detect the Amazon EKS or Kubernetes cluster name.
- **log_group_name**— specifies the log group name for the scraped Prometheus metrics. This field is optional. If you omit it, CloudWatch uses **/aws/containerinsights/*cluster_name*/prometheus** for logs from Amazon EKS and Kubernetes clusters.
- **prometheus_config_path**— specifies the Prometheus scrape configuration file path. If the value of this field starts with `env:` the Prometheus scrape configuration file contents will be retrieved from the container's environment variable. Do not change this field.
- **ecs_service_discovery**— is the section to specify the configuration for Amazon ECS Prometheus service discovery. For more information, see Detailed guide for autodiscovery on Amazon ECS clusters.

  The `ecs_service_discovery` section can contain the following fields:

- `sd_frequency` is the frequency to discover the Prometheus exporters. Specify a number and a unit suffix. For example, `1m` for once per minute or `30s` for once per 30 seconds. Valid unit suffixes are `ns`, `us`, `ms`, `s`, `m`, and `h`.

  This field is optional. The default is 60 seconds (1 minute).
- `sd_target_cluster` is the target Amazon ECS cluster name for auto-discovery. This field is optional. The default is the name of the Amazon ECS cluster where the CloudWatch agent is installed.
- `sd_cluster_region` is the target Amazon ECS cluster's Region. This field is optional. The default is the Region of the Amazon ECS cluster where the CloudWatch agent is installed. .
- `sd_result_file` is the path of the YAML file for the Prometheus target results. The Prometheus scrape configuration will refer to this file.
- `docker_label` is an optional section that you can use to specify the configuration for Docker label-based service discovery. If you omit this section, Docker label-based discovery is not used. This section can contain the following fields:
  - `sd_port_label` is the container's docker label name that specifies the container port for Prometheus metrics. The default value is `ECS_PROMETHEUS_EXPORTER_PORT`. If the container does not have this Docker label, the CloudWatch agent will skip it.
  - `sd_metrics_path_label` is the container's docker label name that specifies the Prometheus metrics path. The default value is `ECS_PROMETHEUS_METRICS_PATH`. If the container does not have this Docker label, the agent assumes the default path `/metrics`.
  - `sd_job_name_label` is the container's docker label name that specifies the Prometheus scrape job name. The default value is `job`. If the container does not have this docker label, the CloudWatch agent uses the job name in the Prometheus scrape configuration.
- `task_definition_list` is an optional section that you can use to specify the configuration of task definition-based service discovery. If you omit this section, task definition-based discovery is not used. This section can contain the following fields:
  - `sd_task_definition_arn_pattern` is the pattern to use to specify the Amazon ECS task definitions to discover. This is a regular expression.
  - `sd_metrics_ports` lists the containerPort for the Prometheus metrics. Separate the containerPorts with semicolons.
  - `sd_container_name_pattern` specifies the Amazon ECS task container names. This is a regular expression.

  - `sd_metrics_path` specifies the Prometheus metric path. If you omit this, the agent assumes the default path `/metrics`
  - `sd_job_name` specifies the Prometheus scrape job name. If you omit this field, the CloudWatch agent uses the job name in the Prometheus scrape configuration.

- **metric_declaration**— are sections that specify the array of logs with embedded metric format to be generated. There are `metric_declaration` sections for each Prometheus source that the CloudWatch agent imports from by default. These sections each include the following fields:
  - `label_matcher` is a regular expression that checks the value of the labels listed in `source_labels`. The metrics that match are enabled for inclusion in the embedded metric format sent to CloudWatch.

    If you have multiple labels specified in `source_labels`, the agent recommend that you do not use ^ or $ characters in the regular expression for `label_matcher`.
  - `source_labels` specifies the value of the labels that are checked by the `label_matcher` line.
  - `label_separator` specifies the separator to be used in the `label_matcher` line if multiple `source_labels` are specified. The default is `;`. You can see this default used in the `label_matcher` line in the following example.
  - `metric_selectors` is a regular expression that specifies the metrics to be collected and sent to CloudWatch.
  - `dimensions` is the list of labels to be used as CloudWatch dimensions for each selected metric.

See the following `metric_declaration` example.

```
"metric_declaration": [
  {
     "source_labels":[ "Service", "Namespace"],
     "label_matcher":"(.*node-exporter.*|.*kube-dns.*);kube-system",
     "dimensions":[
        ["Service", "Namespace"]
     ],
     "metric_selectors":[
        "^coredns_dns_request_type_count_total$"
     ]
  }
```

```
]
```

This example configures an embedded metric format section to be sent as a log event if the following conditions are met:

- The value of `Service` contains either `node-exporter` or `kube-dns`.
- The value of `Namespace` is `kube-system`.
- The Prometheus metric `coredns_dns_request_type_count_total` contains both `Service` and `Namespace` labels.

The log event that is sent includes the following highlighted section:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"coredns_dns_request_type_count_total"
            }
         ],
         "Dimensions":[
            [
               "Namespace",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "Namespace":"kube-system",
   "Service":"kube-dns",
   "coredns_dns_request_type_count_total":2562,
   "eks_amazonaws_com_component":"kube-dns",
   "instance":"192.168.61.254:9153",
   "job":"kubernetes-service-endpoints",
   ...
}
```

Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics

The Kubernetes API Server exposes Prometheus metrics on endpoints by default. The official example for the Kubernetes API Server scraping configuration is available on Github.

The following tutorial shows how to do the following steps to begin importing Kubernetes API Server metrics into CloudWatch:

- Adding the Prometheus scraping configuration for Kubernetes API Server to the CloudWatch agent YAML file.
- Configuring the embedded metric format metrics definitions in the CloudWatch agent YAML file.
- (Optional) Creating a CloudWatch dashboard for the Kubernetes API Server metrics.

> **Note**
>
> The Kubernetes API Server exposes gauge, counter, histogram, and summary metrics. In this release of Prometheus metrics support, CloudWatch imports only the metrics with gauge, counter, and summary types.

To start collecting Kubernetes API Server Prometheus metrics in CloudWatch

1. Download the latest version of the `prometheus-eks.yaml`, `prometheus-eks-fargate.yaml`, or `prometheus-k8s.yaml` file by entering one of the following commands.

   For an Amazon EKS cluster with the EC2 launch type, enter the following command:

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
   ```

   For an Amazon EKS cluster with the Fargate launch type, enter the following command:

   ```
   curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
   ```

   For a Kubernetes cluster running on an Amazon EC2 instance, enter the following command:

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, find the `prometheus-config` section, and add the following section inside of that section. Then save the changes:

```
    # Scrape config for API servers
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
        - role: endpoints
          namespaces:
            names:
              - default
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name]
        action: keep
        regex: kubernetes;https
      - action: replace
        source_labels:
        - __meta_kubernetes_namespace
        target_label: Namespace
      - action: replace
        source_labels:
        - __meta_kubernetes_service_name
        target_label: Service
```

3. While you still have the YAML file open in the text editor, find the `cwagentconfig.json` section. Add the following subsection and save the changes. This section puts the API server metrics onto the CloudWatch agent allow list. Three types of API Server metrics are added to the allow list:

   - etcd object counts
   - API Server registration controller metrics
   - API Server request metrics

```
{"source_labels": ["job", "resource"],
  "label_matcher": "^kubernetes-apiservers;(services|daemonsets.apps|
deployments.apps|configmaps|endpoints|secrets|serviceaccounts|replicasets.apps)",
  "dimensions": [["ClusterName","Service","resource"]],
  "metric_selectors": [
  "^etcd_object_counts$"
  ]
},
{"source_labels": ["job", "name"],
   "label_matcher": "^kubernetes-apiservers;APIServiceRegistrationController$",
   "dimensions": [["ClusterName","Service","name"]],
   "metric_selectors": [
   "^workqueue_depth$",
   "^workqueue_adds_total$",
   "^workqueue_retries_total$"
  ]
},
{"source_labels": ["job","code"],
  "label_matcher": "^kubernetes-apiservers;2[0-9]{2}$",
  "dimensions": [["ClusterName","Service","code"]],
  "metric_selectors": [
   "^apiserver_request_total$"
  ]
},
{"source_labels": ["job"],
  "label_matcher": "^kubernetes-apiservers",
  "dimensions": [["ClusterName","Service"]],
  "metric_selectors": [
  "^apiserver_request_total$"
  ]
},
```

4. If you already have the CloudWatch agent with Prometheus support deployed in the cluster, you must delete it by entering the following command:

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

5. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter:

```
kubectl apply -f prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

For a Kubernetes cluster, enter the following command. Replace *MyCluster* and *region* with values to match your deployment.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

After you have done this, you should see a new log stream named **kubernetes-apiservers** in the **/aws/containerinsights/*cluster_name*/prometheus** log group. This log stream should include log events with an embedded metric format definition like the following:

```
{
   "CloudWatchMetrics":[
      {
         "Metrics":[
            {
               "Name":"apiserver_request_total"
            }
         ],
         "Dimensions":[
            [
               "ClusterName",
               "Service"
            ]
         ],
         "Namespace":"ContainerInsights/Prometheus"
      }
   ],
   "ClusterName":"my-cluster-name",
   "Namespace":"default",
```

```
    "Service":"kubernetes",
    "Timestamp":"1592267020339",
    "Version":"0",
    "apiserver_request_count":0,
    "apiserver_request_total":0,
    "code":"0",
    "component":"apiserver",
    "contentType":"application/json",
    "instance":"192.0.2.0:443",
    "job":"kubernetes-apiservers",
    "prom_metric_type":"counter",
    "resource":"pods",
    "scope":"namespace",
    "verb":"WATCH",
    "version":"v1"
}
```

You can view your metrics in the CloudWatch console in the **ContainerInsights/Prometheus** namespace. You can also optionally create a CloudWatch dashboard for your Prometheus Kubernetes API Server metrics.

(Optional) Creating a dashboard for Kubernetes API Server metrics

To see Kubernetes API Server metrics in your dashboard, you must have first completed the steps in the previous sections to start collecting these metrics in CloudWatch.

To create a dashboard for Kubernetes API Server metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Make sure you have the correct AWS Region selected.
3. In the navigation pane, choose **Dashboards**.
4. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
5. In **Add to this dashboard**, choose **Cancel**.
6. Choose **Actions**, **View/edit source**.
7. Download the following JSON file: Kubernetes API Dashboard source.
8. Open the JSON file that you downloaded with a text editor, and make the following changes:

   - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.

- Replace all the `{{YOUR_AWS_REGION}}` strings with the name of the Region where the metrics are collected. For example `us-west-2`. Be sure not to add whitespaces before or after the text.

9. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
10. Choose **Update**, **Save dashboard**.

(Optional) Set up sample containerized Amazon EKS workloads for Prometheus metric testing

To test the Prometheus metric support in CloudWatch Container Insights, you can set up one or more of the following containerized workloads. The CloudWatch agent with Prometheus support automatically collects metrics from each of these workloads. To see the metrics that are collected by default, see Prometheus metrics collected by the CloudWatch agent.

Before you can install any of these workloads, you must install Helm 3.x by entering the following commands:

```
brew install helm
```

For more information, see Helm.

Topics

- Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes
- Set up NGINX with sample traffic on Amazon EKS and Kubernetes
- Set up memcached with a metric exporter on Amazon EKS and Kubernetes
- Set up Java/JMX sample workload on Amazon EKS and Kubernetes
- Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes
- Tutorial for adding a new Prometheus scrape target: Redis OSS on Amazon EKS and Kubernetes clusters

Set up AWS App Mesh sample workload for Amazon EKS and Kubernetes

Prometheus support in CloudWatch Container Insights supports AWS App Mesh. The following sections explain how to set up App Mesh.

Topics

- Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster
- Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the EC2 launch type or a Kubernetes cluster

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the EC2 launch type, or a Kubernetes cluster.

Configure IAM permissions

You must add the **AWSAppMeshFullAccess** policy to the IAM role for your Amazon EKS or Kubernetes node group. On Amazon EKS, this node group name looks similar to `eksctl-integ-test-eks-prometheus-NodeInstanceRole-ABCDEFHIJKL`. On Kubernetes, it might look similar to `nodes.integ-test-kops-prometheus.k8s.local`.

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in App Mesh Controller.

Install a sample application

aws-app-mesh-examples contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that shows how http routes can use headers for matching incoming requests.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-http-headers.
2. Launch a curler pod to generate traffic:

   ```
   kubectl -n default run -it curler --image=tutum/curl /bin/bash
   ```

3. Curl different endpoints by changing HTTP headers. Run the curl command multiple times, as shown:

   ```
   curl -H "color_header: blue" front.howto-k8s-http-headers.svc.cluster.local:8080/;
    echo;
   ```

```
curl -H "color_header: red" front.howto-k8s-http-headers.svc.cluster.local:8080/;
 echo;

curl -H "color_header: yellow" front.howto-k8s-http-
headers.svc.cluster.local:8080/; echo;
```

4. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-http-headers/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up AWS App Mesh sample workload on an Amazon EKS cluster with the Fargate launch type

Use these instructions if you are setting up App Mesh on a cluster running Amazon EKS with the Fargate launch type.

Configure IAM permissions

To set up IAM permissions, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create iamserviceaccount --cluster MyCluster \
 --namespace howto-k8s-fargate \
 --name appmesh-pod \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
```

```
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
--override-existing-serviceaccounts \
--approve
```

Install App Mesh

To install the App Mesh Kubernetes controller, follow the instructions in App Mesh Controller. Be sure to follow the instructions for Amazon EKS with the Fargate launch type.

Install a sample application

aws-app-mesh-examples contains several Kubernetes App Mesh walkthroughs. For this tutorial, you install a sample color application that works for Amazon EKS clusters with the Fargate launch type.

To use a sample App Mesh application to test Container Insights

1. Install the application using these instructions: https://github.com/aws/aws-app-mesh-examples/tree/main/walkthroughs/howto-k8s-fargate.

   Those instructions assume that you are creating a new cluster with the correct Fargate profile. If you want to use an Amazon EKS cluster that you've already set up, you can use the following commands to set up that cluster for this demonstration. Replace *MyCluster* with the name of your cluster.

   ```
   eksctl create iamserviceaccount --cluster MyCluster \
    --namespace howto-k8s-fargate \
    --name appmesh-pod \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshEnvoyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapDiscoverInstanceAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSCloudMapFullAccess \
    --override-existing-serviceaccounts \
    --approve
   ```

   ```
   eksctl create fargateprofile --cluster MyCluster \
   ```

```
--namespace howto-k8s-fargate --name howto-k8s-fargate
```

2. Port forward the front application deployment:

```
kubectl -n howto-k8s-fargate port-forward deployment/front 8080:8080
```

3. Curl the front app:

```
while true; do  curl -s http://localhost:8080/color; sleep 0.1; echo ; done
```

4. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
5. In the AWS Region where your cluster is running, choose **Metrics** in the navigation pane. The metric are in the **ContainerInsights/Prometheus** namespace.
6. To see the CloudWatch Logs events, choose **Log groups** in the navigation pane. The events are in the log group `/aws/containerinsights/your_cluster_name/prometheus` in the log stream `kubernetes-pod-appmesh-envoy`.

Deleting the App Mesh test environment

When you have finished using App Mesh and the sample application, use the following commands to delete the unnecessary resources. Delete the sample application by entering the following command:

```
cd aws-app-mesh-examples/walkthroughs/howto-k8s-fargate/
kubectl delete -f _output/manifest.yaml
```

Delete the App Mesh controller by entering the following command:

```
helm delete appmesh-controller -n appmesh-system
```

Set up NGINX with sample traffic on Amazon EKS and Kubernetes

NGINX is a web server that can also be used as a load balancer and reverse proxy. For more information about how Kubernetes uses NGINX for ingress , see kubernetes/ingress-nginx.

To install Ingress-NGINX with a sample traffic service to test Container Insights Prometheus support

1. Enter the following command to add the Helm ingress-nginx repo:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

2. Enter the following commands:

```
kubectl create namespace nginx-ingress-sample

helm install my-nginx ingress-nginx/ingress-nginx \
--namespace nginx-ingress-sample \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="10254" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

3. Check whether the services started correctly by entering the following command:

```
kubectl get service -n nginx-ingress-sample
```

   The output of this command should display several columns, including an `EXTERNAL-IP` column.

4. Set an `EXTERNAL-IP` variable to the value of the `EXTERNAL-IP` column in the row of the NGINX ingress controller.

```
EXTERNAL_IP=your-nginx-controller-external-ip
```

5. Start some sample NGINX traffic by entering the following command.

```
SAMPLE_TRAFFIC_NAMESPACE=nginx-sample-traffic
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/nginx-traffic/nginx-traffic-sample.yaml |
sed "s/{{external_ip}}/$EXTERNAL_IP/g" |
sed "s/{{namespace}}/$SAMPLE_TRAFFIC_NAMESPACE/g" |
kubectl apply -f -
```

6. Enter the following command to confirm that all three pods are in the `Running` status.

```
kubectl get pod -n $SAMPLE_TRAFFIC_NAMESPACE
```

   If they are running, you should soon see metrics in the **ContainerInsights/Prometheus** namespace.

To uninstall NGINX and the sample traffic application

1. Delete the sample traffic service by entering the following command:

```
kubectl delete namespace $SAMPLE_TRAFFIC_NAMESPACE
```

2. Delete the NGINX egress by the Helm release name.

```
helm uninstall my-nginx --namespace nginx-ingress-sample
kubectl delete namespace nginx-ingress-sample
```

Set up memcached with a metric exporter on Amazon EKS and Kubernetes

memcached is an open-source memory object caching system. For more information, see What is Memcached?.

If you are running memcached on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace memcached-sample --name memcached-sample
```

To install memcached with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the repo:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace memcached-sample
```

3. Enter the following command to install Memcached

```
helm install my-memcached bitnami/memcached --namespace memcached-sample \
--set metrics.enabled=true \
--set-string serviceAnnotations.prometheus\\.io/port="9150" \
--set-string serviceAnnotations.prometheus\\.io/scrape="true"
```

4. Enter the following command to confirm the annotation of the running service:

```
kubectl describe service my-memcached-metrics -n memcached-sample
```

You should see the following two annotations:

```
Annotations:   prometheus.io/port: 9150
               prometheus.io/scrape: true
```

To uninstall memcached

- Enter the following commands:

```
helm uninstall my-memcached --namespace memcached-sample
kubectl delete namespace memcached-sample
```

Set up Java/JMX sample workload on Amazon EKS and Kubernetes

JMX Exporter is an official Prometheus exporter that can scrape and expose JMX mBeans as Prometheus metrics. For more information, see prometheus/jmx_exporter.

Container Insights can collect predefined Prometheus metrics from Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter.

Default Prometheus scrape configuration

By default, the CloudWatch agent with Prometheus support scrapes the Java/JMX Prometheus metrics from `http://CLUSTER_IP:9404/metrics` on each pod in an Amazon EKS or Kubernetes cluster. This is done by `role:` pod discovery of Prometheus `kubernetes_sd_config`. 9404 is the default port allocated for JMX Exporter by Prometheus. For more information about `role:` pod discovery, see pod. You can configure the JMX Exporter to expose the metrics on a different port or metrics_path. If you do change the port or path, update the default jmx scrape_config in the CloudWatch agent config map. Run the following command to get the current CloudWatch agent Prometheus configuration:

```
kubectl describe cm prometheus-config -n amazon-cloudwatch
```

The fields to change are the `/metrics` and `regex: '.*:9404$'` fields, as highlighted in the following example.

```
job_name: 'kubernetes-jmx-pod'
sample_limit: 10000
metrics_path: /metrics
kubernetes_sd_configs:
- role: pod
relabel_configs:
- source_labels: [__address__]
  action: keep
  regex: '.*:9404$'
- action: replace
  regex: (.+)
  source_labels:
```

Other Prometheus scrape configuration

If you expose your application running on a set of pods with Java/JMX Prometheus exporters by a Kubernetes Service, you can also switch to use `role: service` discovery or `role: endpoint` discovery of Prometheus `kubernetes_sd_config`. For more information about these discovery methods, see service, endpoints, and <kubernetes_sd_config>..

More meta labels are provided by these two service discovery modes which could be useful for you to build the CloudWatch metrics dimensions. For example, you can relabel `__meta_kubernetes_service_name` to `Service` and include it into your metrics' dimension. For more informatio about customizing your CloudWatch metrics and their dimensions, see CloudWatch agent configuration for Prometheus.

Docker image with JMX Exporter

Next, build a Docker image. The following sections provide two example Dockerfiles.

When you have built the image, load it into Amazon EKS or Kubernetes, and then run the following command to verify that Prometheus metrics are exposed by `JMX_EXPORTER` on port 9404. Replace *`$JAR_SAMPLE_TRAFFIC_POD`* with the running pod name and replace *`$JAR_SAMPLE_TRAFFIC_NAMESPACE`* with your application namespace.

If you are running JMX Exporter on a cluster with the Fargate launch type, you also need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *`MyCluster`* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $JAR_SAMPLE_TRAFFIC_NAMESPACE\
 --name $JAR_SAMPLE_TRAFFIC_NAMESPACE
```

```
kubectl exec $JAR_SAMPLE_TRAFFIC_POD -n $JARCAT_SAMPLE_TRAFFIC_NAMESPACE -- curl
 http://localhost:9404
```

Example: Apache Tomcat Docker image with Prometheus metrics

Apache Tomcat server exposes JMX mBeans by default. You can integrate JMX Exporter with Tomcat to expose JMX mBeans as Prometheus metrics. The following example Dockerfile shows the steps to build a testing image:

```
# From Tomcat 9.0 JDK8 OpenJDK
FROM tomcat:9.0-jdk8-openjdk

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter
COPY ./setenv.sh /usr/local/tomcat/bin
COPY your web application.war /usr/local/tomcat/webapps/

RUN chmod  o+x /usr/local/tomcat/bin/setenv.sh

ENTRYPOINT ["catalina.sh", "run"]
```

The following list explains the four COPY lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

  Here is a sample configuration file for Java and Tomcat:

  ```
  lowercaseOutputName: true
  lowercaseOutputLabelNames: true

  rules:
  ```

```
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE

- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
```

```
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `setenv.sh` is a Tomcat startup script to start the JMX exporter along with Tomcat and expose Prometheus metrics on port 9404 of the localhost. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat setenv.sh
export JAVA_OPTS="-javaagent:/opt/jmx_exporter/
jmx_prometheus_javaagent-0.12.0.jar=9404:/opt/jmx_exporter/config.yaml $JAVA_OPTS"
```

- your web application.war is your web application `war` file to be loaded by Tomcat.

Build a Docker image with this configuration and upload it to an image repository.

Example: Java Jar Application Docker image with Prometheus metrics

The following example Dockerfile shows the steps to build a testing image:

```
# Alpine Linux with OpenJDK JRE
FROM openjdk:8-jre-alpine

RUN mkdir -p /opt/jmx_exporter

COPY ./jmx_prometheus_javaagent-0.12.0.jar /opt/jmx_exporter
COPY ./SampleJavaApplication-1.0-SNAPSHOT.jar /opt/jmx_exporter
COPY ./start_exporter_example.sh /opt/jmx_exporter
COPY ./config.yaml /opt/jmx_exporter

RUN chmod -R o+x /opt/jmx_exporter
RUN apk add curl

ENTRYPOINT exec /opt/jmx_exporter/start_exporter_example.sh
```

The following list explains the four `COPY` lines in this Dockerfile.

- Download the latest JMX Exporter jar file from https://github.com/prometheus/jmx_exporter.
- `config.yaml` is the JMX Exporter configuration file. For more information, see https://github.com/prometheus/jmx_exporter#Configuration.

Here is a sample configuration file for Java and Tomcat:

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true

rules:
- pattern: 'java.lang<type=OperatingSystem><>(FreePhysicalMemorySize|
TotalPhysicalMemorySize|FreeSwapSpaceSize|TotalSwapSpaceSize|SystemCpuLoad|
ProcessCpuLoad|OpenFileDescriptorCount|AvailableProcessors)'
  name: java_lang_OperatingSystem_$1
  type: GAUGE

- pattern: 'java.lang<type=Threading><>(TotalStartedThreadCount|ThreadCount)'
  name: java_lang_threading_$1
  type: GAUGE

- pattern: 'Catalina<type=GlobalRequestProcessor, name=\"(\w+-\w+)-(\d+)\"><>(\w+)'
  name: catalina_globalrequestprocessor_$3_total
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina global $3
  type: COUNTER

- pattern: 'Catalina<j2eeType=Servlet, WebModule=//([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-
a-zA-Z0-9+&@#/%=~_|]), name=([-a-zA-Z0-9+/$%~_-|!.]*), J2EEApplication=none,
 J2EEServer=none><>(requestCount|maxTime|processingTime|errorCount)'
  name: catalina_servlet_$3_total
  labels:
    module: "$1"
    servlet: "$2"
  help: Catalina servlet $3 total
  type: COUNTER

- pattern: 'Catalina<type=ThreadPool, name="(\w+-\w+)-(\d+)"><>(currentThreadCount|
currentThreadsBusy|keepAliveCount|pollerThreadCount|connectionCount)'
  name: catalina_threadpool_$3
  labels:
    port: "$2"
    protocol: "$1"
  help: Catalina threadpool $3
  type: GAUGE
```

```
- pattern: 'Catalina<type=Manager, host=([-a-zA-Z0-9+&@#/%?=~_|!:.,;]*[-a-zA-
Z0-9+&@#/%=~_|]), context=([-a-zA-Z0-9+/$%~_-|!.]*)><>(processingTime|sessionCounter|
rejectedSessions|expiredSessions)'
  name: catalina_session_$3_total
  labels:
    context: "$2"
    host: "$1"
  help: Catalina session $3 total
  type: COUNTER

- pattern: ".*"
```

- `start_exporter_example.sh` is the script to start the JAR application with the Prometheus metrics exported. It also provides the JMX Exporter with the `config.yaml` file path.

```
$ cat start_exporter_example.sh
java -javaagent:/opt/jmx_exporter/jmx_prometheus_javaagent-0.12.0.jar=9404:/
opt/jmx_exporter/config.yaml -cp  /opt/jmx_exporter/SampleJavaApplication-1.0-
SNAPSHOT.jar com.gubupt.sample.app.App
```

- SampleJavaApplication-1.0-SNAPSHOT.jar is the sample Java application jar file. Replace it with the Java application that you want to monitor.

Build a Docker image with this configuration and upload it to an image repository.

Set up HAProxy with a metric exporter on Amazon EKS and Kubernetes

HAProxy is an open-source proxy application. For more information, see HAProxy.

If you are running HAProxy on a cluster with the Fargate launch type, you need to set up a Fargate profile before doing the steps in this procedure. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace haproxy-ingress-sample --name haproxy-ingress-sample
```

To install HAProxy with a metric exporter to test Container Insights Prometheus support

1. Enter the following command to add the Helm incubator repo:

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts
```

2. Enter the following command to create a new namespace:

```
kubectl create namespace haproxy-ingress-sample
```

3. Enter the following commands to install HAProxy:

```
helm install haproxy haproxy-ingress/haproxy-ingress \
--namespace haproxy-ingress-sample \
--set defaultBackend.enabled=true \
--set controller.stats.enabled=true \
--set controller.metrics.enabled=true \
--set-string controller.metrics.service.annotations."prometheus\.io/port"="9101" \
--set-string controller.metrics.service.annotations."prometheus\.io/scrape"="true"
```

4. Enter the following command to confirm the annotation of the service:

```
kubectl describe service haproxy-haproxy-ingress-metrics -n haproxy-ingress-sample
```

   You should see the following annotations.

```
Annotations:   prometheus.io/port: 9101
               prometheus.io/scrape: true
```

To uninstall HAProxy

- Enter the following commands:

```
helm uninstall haproxy --namespace haproxy-ingress-sample
kubectl delete namespace haproxy-ingress-sample
```

Tutorial for adding a new Prometheus scrape target: Redis OSS on Amazon EKS and Kubernetes clusters

This tutorial provides a hands-on introduction to scrape the Prometheus metrics of a sample Redis OSS application on Amazon EKS and Kubernetes. Redis OSS (https://redis.io/) is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. For more information, see redis.

redis_exporter (MIT License licensed) is used to expose the Redis OSS Prometheus metrics on the specified port (default: 0.0.0.0:9121). For more information, see redis_exporter.

The Docker images in the following two Docker Hub repositories are used in this tutorial:

- redis
- redis_exporter

To install a sample Redis OSS workload which exposes Prometheus metrics

1. Set the namespace for the sample Redis OSS workload.

```
REDIS_NAMESPACE=redis-sample
```

2. If you are running Redis OSS on a cluster with the Fargate launch type, you need to set up a Fargate profile. To set up the profile, enter the following command. Replace *MyCluster* with the name of your cluster.

```
eksctl create fargateprofile --cluster MyCluster \
--namespace $REDIS_NAMESPACE --name $REDIS_NAMESPACE
```

3. Enter the following command to install the sample Redis OSS workload.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_traffic/redis/redis-traffic-sample.yaml \
| sed "s/{{namespace}}/$REDIS_NAMESPACE/g" \
| kubectl apply -f -
```

4. The installation includes a service named `my-redis-metrics` which exposes the Redis OSS Prometheus metric on port 9121 Enter the following command to get the details of the service:

```
kubectl describe service/my-redis-metrics  -n $REDIS_NAMESPACE
```

   In the `Annotations` section of the results, you'll see two annotations which match the Prometheus scrape configuration of the CloudWatch agent, so that it can auto-discover the workloads:

```
prometheus.io/port: 9121
```

```
prometheus.io/scrape: true
```

The related Prometheus scrape configuration can be found in the `- job_name: kubernetes-service-endpoints` section of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml`.

To start collecting Redis OSS Prometheus metrics in CloudWatch

1. Download the latest version of the of `kubernetes-eks.yaml` or `kubernetes-k8s.yaml` file by entering one of the following commands. For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks.yaml
```

For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-eks-fargate.yaml
```

For a Kubernetes cluster running on an Amazon EC2 instance, enter this command.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-prometheus/prometheus-k8s.yaml
```

2. Open the file with a text editor, and find the `cwagentconfig.json` section. Add the following subsection and save the changes. Be sure that the indentation follows the existing pattern.

```
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName"]],
  "metric_selectors": [
    "^redis_net_(in|out)put_bytes_total$",
    "^redis_(expired|evicted)_keys_total$",
```

```
    "^redis_keyspace_(hits|misses)_total$",
    "^redis_memory_used_bytes$",
    "^redis_connected_clients$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","cmd"]],
  "metric_selectors": [
    "^redis_commands_total$"
  ]
},
{
  "source_labels": ["pod_name"],
  "label_matcher": "^redis-instance$",
  "dimensions": [["Namespace","ClusterName","db"]],
  "metric_selectors": [
    "^redis_db_keys$"
  ]
},
```

The section you added puts the Redis OSS metrics onto the CloudWatch agent allow list. For a list of these metrics, see the following section.

3. If you already have the CloudWatch agent with Prometheus support deployed in this cluster, you must delete it by entering the following command.

```
kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
```

4. Deploy the CloudWatch agent with your updated configuration by entering one of the following commands. Replace *MyCluster* and *region* to match your settings.

   For an Amazon EKS cluster with the EC2 launch type, enter this command.

```
kubectl apply -f prometheus-eks.yaml
```

   For an Amazon EKS cluster with the Fargate launch type, enter this command.

```
cat prometheus-eks-fargate.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

For a Kubernetes cluster, enter this command.

```
cat prometheus-k8s.yaml \
| sed "s/{{cluster_name}}/MyCluster/;s/{{region_name}}/region/" \
| kubectl apply -f -
```

Viewing your Redis OSS Prometheus metrics

This tutorial sends the following metrics to the **ContainerInsights/Prometheus** namespace in CloudWatch. You can use the CloudWatch console to see the metrics in that namespace.

Metric name	Dimensions	
redis_net_input_bytes_total	ClusterName, Namespace	
redis_net_output_bytes_total	ClusterName, Namespace	
redis_expired_keys_total	ClusterName, Namespace	
redis_evicted_keys_total	ClusterName, Namespace	
redis_keyspace_hits_total	ClusterName, Namespace	
redis_keyspace_misses_total	ClusterName, Namespace	

Metric name	Dimensions	
redis_memory_used_bytes	ClusterName, Namespace	
redis_connected_clients	ClusterName, Namespace	
redis_commands_total	ClusterName, Namespace , cmd	
redis_db_keys	ClusterName, Namespace , db	

Note

The value of the **cmd** dimension can be: append, client, command, config, dbsize, flushall, get, incr, info, latency, or slowlog.
The value of the **db** dimension can be db0 to db15.

You can also create a CloudWatch dashboard for your Redis OSS Prometheus metrics.

To create a dashboard for Redis OSS Prometheus metrics

1. Create environment variables, replacing the values below to match your deployment.

```
DASHBOARD_NAME=your_cw_dashboard_name
REGION_NAME=your_metric_region_such_as_us-east-1
CLUSTER_NAME=your_k8s_cluster_name_here
NAMESPACE=your_redis_service_namespace_here
```

2. Enter the following command to create the dashboard.

```
curl https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-
insights/latest/k8s-deployment-manifest-templates/deployment-mode/service/cwagent-
prometheus/sample_cloudwatch_dashboards/redis/cw_dashboard_redis.json \
| sed "s/{{YOUR_AWS_REGION}}/${REGION_NAME}/g" \
| sed "s/{{YOUR_CLUSTER_NAME}}/${CLUSTER_NAME}/g" \
| sed "s/{{YOUR_NAMESPACE}}/${NAMESPACE}/g" \
```

Prometheus metric type conversion by the CloudWatch Agent

The Prometheus client libraries offer four core metric types:

- Counter
- Gauge
- Summary
- Histogram

The CloudWatch agent supports the counter, gauge, and summary metric types.

The Prometheus metrics with the unsupported histogram metric type are dropped by the CloudWatch agent. For more information, see Logging dropped Prometheus metrics.

Gauge metrics

A Prometheus gauge metric is a metric that represents a single numerical value that can arbitrarily go up and down. The CloudWatch agent scrapes gauge metrics and send these values out directly.

Counter metrics

A Prometheus counter metric is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero. The CloudWatch agent calculates a delta from the previous scrape and sends the delta value as the metric value in the log event. So the CloudWatch agent will start to produce one log event from the second scrape and continue with subsequent scrapes, if any.

Summary metrics

A Prometheus summary metric is a complex metric type which is represented by multiple data points. It provides a total count of observations and a sum of all observed values. It calculates configurable quantiles over a sliding time window.

The sum and count of a summary metric are cumulative, but the quantiles are not. The following example shows the variance of quantiles.

```
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 7.123e-06
go_gc_duration_seconds{quantile="0.25"} 9.204e-06
go_gc_duration_seconds{quantile="0.5"} 1.1065e-05
go_gc_duration_seconds{quantile="0.75"} 2.8731e-05
```

```
go_gc_duration_seconds{quantile="1"} 0.003841496
go_gc_duration_seconds_sum 0.37630427
go_gc_duration_seconds_count 9774
```

The CloudWatch agent handles the sum and count of a summary metric in the same way as it handles counter metrics, as described in the previous section. The CloudWatch agent preserves the quantile values as they are originally reported.

Prometheus metrics collected by the CloudWatch agent

The CloudWatch agent with Prometheus support automatically collects metrics from several services and workloads. The metrics that are collected by default are listed in the following sections. You can also configure the agent to collect more metrics from these services, and to collect Prometheus metrics from other applications and services. For more information about collecting additional metrics, see CloudWatch agent configuration for Prometheus.

Prometheus metrics collected from Amazon EKS and Kubernetes clusters are in the **ContainerInsights/Prometheus** namespace. Prometheus metrics collected from Amazon ECS clusters are in the **ECS/ContainerInsights/Prometheus** namespace.

Topics

- Prometheus metrics for App Mesh
- Prometheus metrics for NGINX
- Prometheus metrics for Memcached
- Prometheus metrics for Java/JMX
- Prometheus metrics for HAProxy

Prometheus metrics for App Mesh

The following metrics are automatically collected from App Mesh .

Prometheus metrics for App Mesh on Amazon EKS and Kubernetes clusters

Metric name	Dimensions	
envoy_http_downstream_rq_total	ClusterName, Namespace	

Metric name	Dimensions
envoy_http_downstream_rq_xx	ClusterName, Namespace ClusterName, Namespace , envoy_http_conn_manager_prefix, envoy_response_code_class
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, Namespace
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, Namespace
envoy_cluster_membership_healthy	ClusterName, Namespace
envoy_cluster_membership_total	ClusterName, Namespace
envoy_server_memory_heap_size	ClusterName, Namespace
envoy_server_memory_allocated	ClusterName, Namespace
envoy_cluster_upstream_cx_connect_timeout	ClusterName, Namespace

Metric name	Dimensions	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName, Namespace	
envoy_cluster_upstream_rq_pending_overflow	ClusterName, Namespace	
envoy_cluster_upstream_rq_timeout	ClusterName, Namespace	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName, Namespace	
envoy_cluster_upstream_rq_rx_reset	ClusterName, Namespace	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName, Namespace	

Metric name	Dimensions	
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName, Namespace	
envoy_cluster_upstream_rq_maintenance_mode	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, Namespace	
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, Namespace	

Metric name	Dimensions
envoy_cluster_upstream_flow_control_drained_total	ClusterName, Namespace
envoy_cluster_upstream_rq_retry	ClusterName, Namespace
envoy_cluster_upstream_rq_retry_success	ClusterName, Namespace
envoy_cluster_upstream_rq_retry_overflow	ClusterName, Namespace
envoy_server_live	ClusterName, Namespace
envoy_server_uptime	ClusterName, Namespace

Prometheus metrics for App Mesh on Amazon ECS clusters

Metric name	Dimensions
envoy_http_downstream_rq_total	ClusterName, TaskDefinitionFamily

Metric name	Dimensions	
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_rx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_tx_bytes_total	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_healthy	ClusterName, TaskDefinitionFamily	
envoy_cluster_membership_total	ClusterName, TaskDefinitionFamily	
envoy_server_memory_heap_size	ClusterName, TaskDefinitionFamily	
envoy_server_memory_allocated	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_connect_timeout	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions	
envoy_cluster_upstream_rq_pending_failure_eject	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_pending_overflow	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_try_per_timeout	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_rq_rx_reset	ClusterName, TaskDefinitionFamily	
envoy_cluster_upstream_cx_destroy_local_with_active_rq	ClusterName, TaskDefinitionFamily	

Metric name	Dimensions
envoy_cluster_upstream_cx_destroy_remote_active_rq	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_maintenance_mode	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_flow_control_paused_reading_total	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_flow_control_resumed_reading_total	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_flow_control_backed_up_total	ClusterName, TaskDefinitionFamily

Metric name	Dimensions
envoy_cluster_upstream_flow_control_drained_total	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_success	ClusterName, TaskDefinitionFamily
envoy_cluster_upstream_rq_retry_overflow	ClusterName, TaskDefinitionFamily
envoy_server_live	ClusterName, TaskDefinitionFamily
envoy_server_uptime	ClusterName, TaskDefinitionFamily
envoy_http_downstream_rq_xx	ClusterName, TaskDefinitionFamily, envoy_http_conn_manager_prefix, envoy_response_code_class ClusterName, TaskDefinitionFamily, envoy_response_code_class

Note

TaskDefinitionFamily is the Kubernetes namespace of the mesh.

The value of envoy_http_conn_manager_prefix can be ingress, egress, or admin. The value of envoy_response_code_class can be 1 (stands for 1xx), 2 stands for 2xx), 3 stands for 3xx), 4 stands for 4xx), or 5 stands for 5xx).

Prometheus metrics for NGINX

The following metrics are automatically collected from NGINX on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions	
nginx_ingress_controller_nginx_process_cpu_seconds_total	ClusterName, Namespace , Service	
nginx_ingress_controller_success	ClusterName, Namespace , Service	
nginx_ingress_controller_requests	ClusterName, Namespace , Service	
nginx_ingress_controller_nginx_process_connections	ClusterName, Namespace , Service	
nginx_ingress_controller_nginx_proce	ClusterName, Namespace , Service	

Metric name	Dimensions	
ss_connections_total		
nginx_ingress_controller_nginx_process_resident_memory_bytes	ClusterName, Namespace , Service	
nginx_ingress_controller_config_last_reload_successful	ClusterName, Namespace , Service	
nginx_ingress_controller_requests	ClusterName, Namespace , Service, status	

Prometheus metrics for Memcached

The following metrics are automatically collected from Memcached on Amazon EKS and Kubernetes clusters.

Metric name	Dimensions	
memcached_current_items	ClusterName, Namespace , Service	

Metric name	Dimensions
memcached_current_connections	ClusterName, Namespace , Service
memcached_limit_bytes	ClusterName, Namespace , Service
memcached_current_bytes	ClusterName, Namespace , Service
memcached_written_bytes_total	ClusterName, Namespace , Service
memcached_read_bytes_total	ClusterName, Namespace , Service
memcached_items_evicted_total	ClusterName, Namespace , Service
memcached_items_reclaimed_total	ClusterName, Namespace , Service
memcached_commands_total	ClusterName, Namespace , Service ClusterName, Namespace , Service, command ClusterName, Namespace , Service, status, command

Prometheus metrics for Java/JMX

Metrics collected on Amazon EKS and Kubernetes clusters

On Amazon EKS and Kubernetes clusters, Container Insights can collect the following predefined Prometheus metrics from the Java Virtual Machine (JVM), Java, and Tomcat (Catalina) using the JMX Exporter. For more information, see prometheus/jmx_exporter on Github.

Java/JMX on Amazon EKS and Kubernetes clusters

Metric name	Dimensions	
jvm_classes_loaded	ClusterName ,Namespace	
jvm_threads_current	ClusterName ,Namespace	
jvm_threads_daemon	ClusterName ,Namespace	
java_lang_operatingsystem_totalswapspacesize	ClusterName ,Namespace	
java_lang_operatingsystem_systemcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_processcpuload	ClusterName ,Namespace	
java_lang_operatingsystem_freeswapspacesize	ClusterName ,Namespace	

Metric name	Dimensions	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName ,Namespace	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName ,Namespace	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName ,Namespace	
java_lang_operatingsystem_availableprocessors	ClusterName ,Namespace	
jvm_memory_bytes_used	ClusterName ,Namespace ,area	
jvm_memory_pool_bytes_used	ClusterName ,Namespace ,pool	

Note

The values of the `area` dimension can be heap or nonheap.

The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon EKS and Kubernetes clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload.

Metric name	Dimensions	
catalina_manager_activesessions	ClusterName ,Namespace	
catalina_manager_rejectedsessions	ClusterName ,Namespace	
catalina_globalrequestprocessor_bytesreceived	ClusterName ,Namespace	
catalina_globalrequestprocessor_bytessent	ClusterName ,Namespace	
catalina_globalrequestprocessor_requestcount	ClusterName ,Namespace	

Metric name	Dimensions	
catalina_globalrequestprocessor_errorcount	ClusterName ,Namespace	
catalina_globalrequestprocessor_processingtime	ClusterName ,Namespace	

Java/JMX on Amazon ECS clusters

Metric name	Dimensions	
jvm_classes_loaded	ClusterName ,TaskDefinitionFamily	
jvm_threads_current	ClusterName ,TaskDefinitionFamily	
jvm_threads_daemon	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_totalswapspacesize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_systemcpuload	ClusterName ,TaskDefinitionFamily	

Metric name	Dimensions	
java_lang_operatingsystem_processcpuload	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_freeswapspacesize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_totalphysicalmemorysize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_freephysicalmemorysize	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_openfiledescriptorcount	ClusterName ,TaskDefinitionFamily	
java_lang_operatingsystem_availableprocessors	ClusterName ,TaskDefinitionFamily	

Metric name	Dimensions	
jvm_memory_bytes_used	ClusterName , TaskDefinitionFamily, area	
jvm_memory_pool_bytes_used	ClusterName , TaskDefinitionFamily, pool	

> **Note**
>
> The values of the area dimension can be heap or nonheap.
> The values of the pool dimension can be Tenured Gen, Compress Class Space, Survivor Space, Eden Space, Code Cache, or Metaspace.

Tomcat/JMX on Amazon ECS clusters

In addition to the Java/JMX metrics in the previous table, the following metrics are also collected for the Tomcat workload on Amazon ECS clusters.

Metric name	Dimensions	
catalina_manager_activesessions	ClusterName , TaskDefinitionFamily	
catalina_manager_rejectedsessions	ClusterName , TaskDefinitionFamily	
catalina_globalrequestprocessor_bytesreceived	ClusterName , TaskDefinitionFamily	

Metric name	Dimensions	
catalina_globalrequestprocessor_bytessent	ClusterName ,TaskDefinitionFamily	
catalina_globalrequestprocessor_requestcount	ClusterName ,TaskDefinitionFamily	
catalina_globalrequestprocessor_errorcount	ClusterName ,TaskDefinitionFamily	
catalina_globalrequestprocessor_processingtime	ClusterName ,TaskDefinitionFamily	

Prometheus metrics for HAProxy

The following metrics are automatically collected from HAProxy on Amazon EKS and Kubernetes clusters.

The metrics collected depend on which version of HAProxy Ingress that you are using. For more information about HAProxy Ingress and its versions, see haproxy-ingress.

Metric name	Dimensions	Availability
haproxy_backend_bytes_in_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_bytes_out_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connection_errors_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_connections_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_backend_http_responses_total	ClusterName , Namespace , Service, code, backend	All versions of HAProxy Ingress
haproxy_backend_status	ClusterName , Namespace , Service	Only in versions 0.10 or later of HAProxy Ingress
haproxy_backend_up	ClusterName , Namespace , Service	Only in versions of HAProxy Ingress earlier than 0.10

Metric name	Dimensions	Availability
haproxy_frontend_bytes_in_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_bytes_out_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_connections_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_current_sessions	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_http_requests_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress
haproxy_frontend_http_responses_total	ClusterName , Namespace , Service, code, frontend	All versions of HAProxy Ingress
haproxy_frontend_request_errors_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Metric name	Dimensions	Availability
haproxy_frontend_requests_denied_total	ClusterName , Namespace , Service	All versions of HAProxy Ingress

Note

The values of the code dimension can be 1xx, 2xx, 3xx, 4xx, 5xx, or other.
The values of the backend dimension can be:

- http-default-backend, http-shared-backend, or httpsback-shared-backend for HAProxy Ingress version 0.0.27 or earlier.
- _default_backend for HAProxy Ingress versions later than 0.0.27.

The values of the frontend dimension can be:

- httpfront-default-backend, httpfront-shared-frontend, or httpfronts for HAProxy Ingress version 0.0.27 or earlier.
- _front_http or _front_https for HAProxy Ingress versions later than 0.0.27.

Viewing your Prometheus metrics

You can monitor and alarm on all your Prometheus metrics including the curated pre-aggregated metrics from App Mesh, NGINX, Java/JMX, Memcached, and HAProxy, and any other manually configured Prometheus exporter you may have added. For more information about collecting metrics from other Prometheus exporters, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

In the CloudWatch console, Container Insights provides the following pre-built reports:

- For Amazon EKS and Kubernetes clusters, there are pre-built reports for App Mesh, NGINX, HAPROXY, Memcached, and Java/JMX.
- For Amazon ECS clusters, there are pre-built reports for App Mesh and Java/JMX.

Container Insights also provides custom dashboards for each of the workloads that Container Insights collects curated metrics from. You can download these dashboards from GitHub

To see all your Prometheus metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the list of namespaces, choose **ContainerInsights/Prometheus** or **ECS/ContainerInsights/Prometheus**.
4. Choose one of the sets of dimensions in the following list. Then select the checkbox next to the metrics that you want to see.

To see pre-built reports on your Prometheus metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Performance Monitoring**.
3. In the drop-down box near the top of the page, choose any of the Prometheus options.

   In the other drop-down box, choose a cluster to view

AWS also provides custom dashboards for NGINX, App Mesh, Memcached, HAProxy, and Java/JMX.

To use a custom dashboard that Amazon has provided

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Dashboards**.
3. Choose **Create Dashboard**. Enter a name for the new dashboard, and choose **Create dashboard**.
4. In **Add to this dashboard**, choose **Cancel**.
5. Choose **Actions**, **View/edit source**.
6. Download one of the following JSON files:

   - NGINX custom dashboard source on Github.
   - App Mesh custom dashboard source on Github.
   - Memcached custom dashboard source on Github
   - HAProxy-Ingress custom dashboard source on Github

- Java/JMX custom dashboard source on Github.

7. Open the JSON file that you downloaded with a text editor, and make the following changes:

   - Replace all the `{{YOUR_CLUSTER_NAME}}` strings with the exact name of your cluster. Make sure not to add whitespaces before or after the text.
   - Replace all the `{{YOUR_REGION}}` strings with the AWS Region where your cluster is running. For example, **`us-west-1`** Make sure not to add whitespaces before or after the text.
   - Replace all the `{{YOUR_NAMESPACE}}` strings with the exact namespace of your workload.
   - Replace all the `{{YOUR_SERVICE_NAME}}` strings with the exact service name of your workload. For example, **`haproxy-haproxy-ingress-controller-metrics`**

8. Copy the entire JSON blob and paste it into the text box in the CloudWatch console, replacing what is already in the box.
9. Choose **Update**, **Save dashboard**.

Prometheus metrics troubleshooting

This section provides help for troubleshooting your Prometheus metrics setup.

Topics

- Prometheus metrics troubleshooting on Amazon ECS
- Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters

Prometheus metrics troubleshooting on Amazon ECS

This section provides help for troubleshooting your Prometheus metrics setup on Amazon ECS clusters.

I don't see Prometheus metrics sent to CloudWatch Logs

The Prometheus metrics should be ingested as log events in the log group **/aws/ecs/containerinsights/cluster-name/Prometheus**. If the log group is not created or the Prometheus metrics are not sent to the log group, you will need to first check whether the Prometheus targets have been successfully discovered by the CloudWatch agent. And next check the security group and permission settings of the CloudWatch agent. The following steps guide you to do the debugging.

Step 1: Enable the CloudWatch agent debugging mode

First, change the CloudWatch agent to debug mode by adding the following bold lines to your CloudFormation template file, `cwagent-ecs-prometheus-metric-for-bridge-host.yaml` or `cwagent-ecs-prometheus-metric-for-awsvpc.yaml`. Then save the file.

```
cwagentconfig.json: |
    {
      "agent": {
        "debug": true
      },
      "logs": {
        "metrics_collected": {
```

Create a new CloudFormation changeset against the existing stack. Set other parameters in the changeset to the same values as in your existing CloudFormation stack. The following example is for a CloudWatch agent installed in an Amazon ECS cluster using the EC2 launch type and the bridge network mode.

```
ECS_NETWORK_MODE=bridge
 CREATE_IAM_ROLES=True
ECS_TASK_ROLE_NAME=your_selected_ecs_task_role_name
ECS_EXECUTION_ROLE_NAME=your_selected_ecs_execution_role_name
NEW_CHANGESET_NAME=your_selected_ecs_execution_role_name

aws cloudformation create-change-set --stack-name CWAgent-Prometheus-ECS-
${ECS_CLUSTER_NAME}-EC2-${ECS_NETWORK_MODE} \
    --template-body file://cwagent-ecs-prometheus-metric-for-bridge-host.yaml \
    --parameters ParameterKey=ECSClusterName,ParameterValue=$ECS_CLUSTER_NAME \
                 ParameterKey=CreateIAMRoles,ParameterValue=$CREATE_IAM_ROLES \
                 ParameterKey=ECSNetworkMode,ParameterValue=$ECS_NETWORK_MODE \
                 ParameterKey=TaskRoleName,ParameterValue=$ECS_TASK_ROLE_NAME \
                 ParameterKey=ExecutionRoleName,ParameterValue=$ECS_EXECUTION_ROLE_NAME
 \
    --capabilities CAPABILITY_NAMED_IAM \
    --region $AWS_REGION \
    --change-set-name $NEW_CHANGESET_NAME
```

Go to the CloudFormation console to review the new changeset, `$NEW_CHANGESET_NAME`. There should be one change applied to the **CWAgentConfigSSMParameter** resource. Execute the changeset and restart the CloudWatch agent task by entering the following commands.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
```

```
--desired-count 0 \
--service your_service_name_here \
--region $AWS_REGION
```

Wait about 10 seconds and then enter the following command.

```
aws ecs update-service --cluster $ECS_CLUSTER_NAME \
--desired-count 1 \
--service your_service_name_here \
--region $AWS_REGION
```

Step 2: Check the ECS service discovery logs

The ECS task definition of the CloudWatch agent enables the logs by default in the section below. The logs are sent to CloudWatch Logs in the log group **/ecs/ecs-cwagent-prometheus**.

```
LogConfiguration:
  LogDriver: awslogs
    Options:
      awslogs-create-group: 'True'
      awslogs-group: "/ecs/ecs-cwagent-prometheus"
      awslogs-region: !Ref AWS::Region
      awslogs-stream-prefix: !Sub 'ecs-${ECSLaunchType}-awsvpc'
```

Filter the logs by the string `ECS_SD_Stats` to get the metrics related to the ECS service discovery, as shown in the following example.

```
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeContainerInstances: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeInstancesRequest: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_DescribeTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: AWSCLI_ListTasks: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: Exporter_DiscoveredTargetCount: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_EC2MetaData: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Get_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_ContainerInstance: 1
2020-09-1T01:53:14Z D! ECS_SD_Stats: LRUCache_Size_TaskDefinition: 2
2020-09-1T01:53:14Z D! ECS_SD_Stats: Latency: 43.399783ms
```

The meaning of each metric for a particular ECS service discovery cycle is as follows:

- **AWSCLI_DescribeContainerInstances** – the number of `ECS::DescribeContainerInstances` API calls made.
- **AWSCLI_DescribeInstancesRequest** – the number of `ECS::DescribeInstancesRequest` API calls made.
- **AWSCLI_DescribeTaskDefinition** – the number of `ECS::DescribeTaskDefinition` API calls made.
- **AWSCLI_DescribeTasks** – the number of `ECS::DescribeTasks` API calls made.
- **AWSCLI_ListTasks** – the number of `ECS::ListTasks` API calls made.
- **ExporterDiscoveredTargetCount** – the number of Prometheus targets that were discovered and successfully exported into the target result file within the container.
- **LRUCache_Get_EC2MetaData** – the number of times that container instances metadata was retrieved from the cache.
- **LRUCache_Get_TaskDefinition** – the number of times that ECS task definition metadata was retrieved from the cache.
- **LRUCache_Size_ContainerInstance** – the number of unique container instance's metadata cached in memory.
- **LRUCache_Size_TaskDefinition** – the number of unique ECS task definitions cached in memory.
- **Latency** – how long the service discovery cycle takes.

Check the value of `ExporterDiscoveredTargetCount` to see whether the discovered Prometheus targets match your expectations. If not, the possible reasons are as follows:

- The configuration of ECS service discovery might not match your application's setting. For the Docker label-based service discovery, your target containers may not have the necessary Docker label configured in the CloudWatch agent to auto discover them. For the ECS task definition ARN regular expression-based service discovery, the regex setting in the CloudWatch agent may not match your application's task definition.
- The CloudWatch agent's ECS task role might not have permission to retrieve the metadata of ECS tasks. Check that the CloudWatch agent has been granted the following read-only permissions:
  - `ec2:DescribeInstances`
  - `ecs:ListTasks`
  - `ecs:DescribeContainerInstances`
  - `ecs:DescribeTasks`

- `ecs:DescribeTaskDefinition`

Step 3: Check the network connection and the ECS task role policy

If there are still no log events sent to the target CloudWatch Logs log group even though the value of `Exporter_DiscoveredTargetCount` indicates that there are discovered Prometheus targets, this could be caused by one of the following:

- The CloudWatch agent might not be able to connect to the Prometheus target ports. Check the security group setting behind the CloudWatch agent. The private IP should alow the CloudWatch agent to connect to the Prometheus exporter ports.
- The CloudWatch agent's ECS task role might not have the **CloudWatchAgentServerPolicy** managed policy. The CloudWatch agent's ECS task role needs to have this policy to be able to send the Prometheus metrics as log events. If you used the sample CloudFormation template to create the IAM roles automatically, both the ECS task role and the ECS execution role are granted with the least privilege to perform the Prometheus monitoring.

Prometheus metrics troubleshooting on Amazon EKS and Kubernetes clusters

This section provides help for troubleshooting your Prometheus metrics setup on Amazon EKS and Kubernetes clusters.

General troubleshooting steps on Amazon EKS

To confirm that the CloudWatch agent is running, enter the following command.

```
kubectl get pod -n amazon-cloudwatch
```

The output should include a row with `cwagent-prometheus-`*`id`* in the `NAME` column and `Running` in the `STATUS column.`

To display details about the running pod, enter the following command. Replace *`pod-name`* with the complete name of your pod that has a name that starts with `cw-agent-prometheus`.

```
kubectl describe pod pod-name -n amazon-cloudwatch
```

If you have CloudWatch Container Insights installed, you can use CloudWatch Logs Insights to query the logs from the CloudWatch agent collecting the Prometheus metrics.

To query the application logs

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the log group for the application logs, **/aws/containerinsights/*cluster-name*/application**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields ispresent(kubernetes.pod_name) as haskubernetes_pod_name, stream,
 kubernetes.pod_name, log |
filter haskubernetes_pod_name and kubernetes.pod_name like /cwagent-prometheus
```

You can also confirm that Prometheus metrics and metadata are being ingested as CloudWatch Logs events.

To confirm that Prometheus data is being ingested

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **CloudWatch Logs Insights**.
3. Select the **/aws/containerinsights/*cluster-name*/prometheus**
4. Replace the search query expression with the following query, and choose **Run query**

```
fields @timestamp, @message | sort @timestamp desc | limit 20
```

Logging dropped Prometheus metrics

This release does not collect Prometheus metrics of the histogram type. You can use the CloudWatch agent to check whether any Prometheus metrics are being dropped because they are histogram metrics. You can also log a list of the first 500 Prometheus metrics that are dropped and not sent to CloudWatch because they are histogram metrics.

To see whether any metrics are being dropped, enter the following command:

```
kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
```

If any metrics are being dropped, you will see the following lines in the `/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log` file.

```
I! Drop Prometheus metrics with unsupported types. Only Gauge, Counter and Summary are
 supported.
I! Please enable CWAgent debug mode to view the first 500 dropped metrics
```

If you see those lines and want to know what metrics are being dropped, use the following steps.

To log a list of dropped Prometheus metrics

1. Change the CloudWatch agent to debug mode by adding the following bold lines to your `prometheus-eks.yaml` or `prometheus-k8s.yaml`file, and save the file.

   ```
   {
         "agent": {
           "debug": true
         },
   ```

   This section of the file should then look like this:

   ```
   cwagentconfig.json: |
       {
         "agent": {
           "debug": true
         },
         "logs": {
           "metrics_collected": {
   ```

2. Reinstall the CloudWatch agent to enable debug mode by entering the following commands:

   ```
   kubectl delete deployment cwagent-prometheus -n amazon-cloudwatch
   kubectl apply -f prometheus.yaml
   ```

   The dropped metrics are logged in the CloudWatch agent pod.

3. To retrieve the logs from the CloudWatch agent pod, enter the following command:

   ```
   kubectl logs -l "app=cwagent-prometheus" -n amazon-cloudwatch --tail=-1
   ```

Or, if you have Container Insights Fluentd logging installed, the logs are also saved in the CloudWatch Logs log group **/aws/containerinsights/*cluster_name*/application**.

To query these logs, you can follow the steps for querying the application logs in General troubleshooting steps on Amazon EKS.

Where are the Prometheus metrics ingested as CloudWatch Logs log events?

The CloudWatch agent creates a log stream for each Prometheus scrape job configuration. For example, in the `prometheus-eks.yaml` and `prometheus-k8s.yaml` files, the line `job_name: 'kubernetes-pod-appmesh-envoy'` scrapes App Mesh metrics. The Prometheus target is defined as `kubernetes-pod-appmesh-envoy`. So all App Mesh Prometheus metrics are ingested as CloudWatch Logs events in the log stream **kubernetes-pod-appmesh-envoy** under the log group named **/aws/containerinsights/cluster-name/Prometheus**.

I don't see Amazon EKS or Kubernetes Prometheus metrics in CloudWatch metrics

First, make sure that the Prometheus metrics are ingested as log events in the log group **/aws/containerinsights/cluster-name/Prometheus**. Use the information in Where are the Prometheus metrics ingested as CloudWatch Logs log events? to help you check the target log stream. If the log stream is not created or there are no new log events in the log stream, check the following:

- Check that the Prometheus metrics exporter endpoints are set up correctly
- Check that the Prometheus scraping configurations in the `config map: cwagent-prometheus` section of the CloudWatch agent YAML file is correct. The configuration should be the same as it would be in a Prometheus configuration file. For more information, see <scrape_config> in the Prometheus documentation.

If the Prometheus metrics are ingested as log events correctly, check that the embedded metric format settings are added into the log events to generate the CloudWatch metrics.

```
"CloudWatchMetrics":[
   {
      "Metrics":[
         {
            "Name":"envoy_http_downstream_cx_destroy_remote_active_rq"
         }
      ],
```

```
      "Dimensions":[
         [
            "ClusterName",
            "Namespace"
         ]
      ],
      "Namespace":"ContainerInsights/Prometheus"
   }
],
```

For more information about embedded metric format, see Specification: Embedded metric format.

If there is no embedded metric format in the log events, check that the `metric_declaration` section is configured correctly in the `config map: prometheus-cwagentconfig` section of the CloudWatch agent installation YAML file. For more information, see Tutorial for adding a new Prometheus scrape target: Prometheus API Server metrics.

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see Detect common application problems with CloudWatch Application Insights.

You can enable Application Insights to gather additional data from your containerized applications and microservices. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** below the performance view in the Container Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your containerized applications, the Application Insights dashboard appears below the Container Insights dashboard.

For more information about Application Insights and containerized applications, see Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring.

Viewing Amazon ECS lifecycle events within Container Insights

You can view Amazon ECS lifecycle events within the Container Insights console. This helps you correlate your container metrics, logs, and events in a single view to give you a more complete operational visibility.

The events include container instance state change events, task state change events, and service action events. They are automatically sent by Amazon ECS to Amazon EventBridge and are also

collected in CloudWatch in event log format. For more information about these events, see Amazon ECS events.

Standard Container Insights pricing applies for Amazon ECS Lifecycle events. For more information, see Amazon CloudWatch Pricing.

To configure the table of lifecycle events and create rules for a cluster, you must have the `events:PutRule`, `events:PutTargets`, and `logs:CreateLogGroup` permissions. You must also make sure that there is a resource policy that enables EventBridge to create the log stream and send logs to CloudWatch Logs. If this resource policy doesn't exist, you can enter the following command to create it:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Principal": {
        "Service": ["events.amazonaws.com", "delivery.logs.amazonaws.com"]
      },
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:/aws/events/ecs/
containerinsights/*:*",
      "Condition": {
        "StringEquals": {
        "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
        "aws:SourceArn": "arn:aws:events:us-east-1:111122223333:rule/
eventsToLog*"
        }
      },
      "Sid": "TrustEventBridgeToStoreECSLifecycleLogEvents"
    }
  ]
}
```

You can use the following command to check whether you already have this policy, and to confirm that attaching it worked correctly.

```
aws logs describe-resource-policies --region region --output json
```

To view the table of lifecycle events, you must have the `events:DescribeRule`, `events:ListTargetsByRule`, and `logs:DescribeLogGroups` permissions.

To view Amazon ECS lifecycle events in the CloudWatch Container Insights console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**, **Container Insights**.
3. Choose **View performance dashboards**.
4. In the next drop-down, choose either **ECS Clusters**, **ECS Services**, or **ECS Tasks**.
5. If you chose **ECS Services** or **ECS Tasks** in the previous step, choose the **Lifecycle events** tab.
6. At the bottom of the page, if you see **Configure lifecycle events**, choose it to create EventBridge rules for your cluster.

   The events are displayed below the container insights panes and above the Application Insights section. To run extra analytics and create additional visualizations on these events, choose **View in Logs Insights** in the Lifecycle Events table.

Troubleshooting Container Insights

The following sections can help if you're having trouble issues with Container Insights.

Failed deployment on Amazon EKS or Kubernetes

If the agent doesn't deploy correctly on a Kubernetes cluster, try the following:

- Run the following command to get the list of pods.

  ```
  kubectl get pods -n amazon-cloudwatch
  ```

- Run the following command and check the events at the bottom of the output.

  ```
  kubectl describe pod pod-name -n amazon-cloudwatch
  ```

- Run the following command to check the logs.

```
kubectl logs pod-name -n amazon-cloudwatch
```

Unauthorized panic: Cannot retrieve cadvisor data from kubelet

If your deployment fails with the error `Unauthorized panic: Cannot retrieve cadvisor data from kubelet`, your kubelet might not have Webhook authorization mode enabled. This mode is required for Container Insights. For more information, see Setup guide (AWS CLI).

Deploying Container Insights on a deleted and re-created cluster on Amazon ECS

If you delete an existing Amazon ECS cluster that does not have Container Insights enabled, and you re-create it with the same name, you can't enable Container Insights on this new cluster at the time you re-create it. You can enable it by re-creating it, and then entering the following command:

```
aws ecs update-cluster-settings --cluster myCICluster --settings name=container
 Insights,value=enabled
```

Invalid endpoint error

If you see an error message similar to the following, check to make sure that you replaced all the placeholders such as *cluster-name* and *region-name* in the commands that you are using with the correct information for your deployment.

```
"log": "2020-04-02T08:36:16Z E! cloudwatchlogs: code: InvalidEndpointURL, message:
 invalid endpoint uri, original error: &url.Error{Op:\"parse\", URL:\"https://
logs.{{region_name}}.amazonaws.com/\", Err:\"{\"}, &awserr.baseError{code:
\"InvalidEndpointURL\", message:\"invalid endpoint uri\", errs:[]error{(*url.Error)
(0xc0008723c0)}}\n",
```

Metrics don't appear in the console

If you don't see any Container Insights metrics in the AWS Management Console, be sure that you have completed the setup of Container Insights. Metrics don't appear before Container Insights has been set up completely. For more information, see Container Insights.

Pod metrics missing on Amazon EKS or Kubernetes after upgrading cluster

This section might be useful if all or some pod metrics are missing after you deploy the CloudWatch agent as a daemonset on a new or upgraded cluster, or you see an error log with the message `W! No pod metric collected`.

These errors can be caused by changes in the container runtime, such as containerd or the Docker systemd cgroup driver. You can usually solve this by updating your deployment manifest so that the containerd socket from the host is mounted into the container. See the following example:

```
# For full example see https://github.com/aws-samples/amazon-cloudwatch-container-
insights/blob/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/
container-insights-monitoring/cwagent/cwagent-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  template:
    spec:
      containers:
        - name: cloudwatch-agent
# ...
          # Don't change the mountPath
          volumeMounts:
# ...
            - name: dockersock
              mountPath: /var/run/docker.sock
              readOnly: true
            - name: varlibdocker
              mountPath: /var/lib/docker
              readOnly: true
            - name: containerdsock # NEW mount
              mountPath: /run/containerd/containerd.sock
              readOnly: true
# ...
      volumes:
# ...
        - name: dockersock
          hostPath:
            path: /var/run/docker.sock
        - name: varlibdocker
```

```
          hostPath:
            path: /var/lib/docker
        - name: containerdsock # NEW volume
          hostPath:
            path: /run/containerd/containerd.sock
```

No pod metrics when using Bottlerocket for Amazon EKS

Bottlerocket is a Linux-based open source operating system that is purpose-built by AWS for running containers.

Bottlerocket uses a different `containerd` path on the host, so you need to change the volumes to its location. If you don't, you see an error in the logs that includes `W! No pod metric collected`. See the following example.

```
volumes:
  # ...
    - name: containerdsock
      hostPath:
        # path: /run/containerd/containerd.sock
        # bottlerocket does not mount containerd sock at normal place
        # https://github.com/bottlerocket-os/bottlerocket/
commit/91810c85b83ff4c3660b496e243ef8b55df0973b
        path: /run/dockershim.sock
```

No container filesystem metrics when using the containerd runtime for Amazon EKS or Kubernetes

This is a known issue and is being worked on by community contributors. For more information, see Disk usage metric for containerd and container file system metrics is not supported by cadvisor for containerd on GitHub.

Unexpected log volume increase from CloudWatch agent when collecting Prometheus metrics

This was a regression introduced in version 1.247347.6b250880 of the CloudWatch agent. This regression has already been fixed in more recent versions of the agent. It's impact was limited to scenarios where customers collected the logs of the CloudWatch agent itself and were also using Prometheus. For more information, see [prometheus] agent is printing all the scraped metrics in log on GitHub.

Latest docker image mentioned in release notes not found from Dockerhub

The release note and tag on Github are updated before the actual release internally. It usually takes 1-2 weeks to see the latest Docker image on registries after the version number on Github is bumped. There is no nightly release for the CloudWatch agent container image. You can build the image directly from source at the following location: https://github.com/aws/amazon-cloudwatch-agent/tree/main/amazon-cloudwatch-container-insights/cloudwatch-agent-dockerfile

CrashLoopBackoff error on the CloudWatch agent

If you see a `CrashLoopBackOff` error for the CloudWatch agent, make sure that your IAM permissions are set correctly. For more information, see Setup guide (AWS CLI).

CloudWatch agent or Fluentd pod stuck in pending

If you have a CloudWatch agent or Fluentd pod stuck in `Pending` or with a `FailedScheduling` error, determine if your nodes have enough compute resources based on the number of cores and amount of RAM required by the agents. Enter the following command to describe the pod:

```
kubectl describe pod cloudwatch-agent-85ppg -n amazon-cloudwatch
```

Building your own CloudWatch agent Docker image

You can build your own CloudWatch agent Docker image by referring to the Dockerfile located at https://github.com/aws-samples/amazon-cloudwatch-container-insights/blob/latest/cloudwatch-agent-dockerfile/Dockerfile.

The Dockerfile supports building multi-architecture images directly using `docker buildx`.

Deploying other CloudWatch agent features in your containers

You can deploy additional monitoring features in your containers using the CloudWatch agent. These features include the following:

- **Embedded Metric Format**— For more information, see Embedding metrics within logs.
- **StatsD**— For more information, see Retrieve custom metrics with StatsD.

Instructions and necessary files are located on GitHub at the following locations:

- For Amazon ECS containers, see Example Amazon ECS task definitions based on deployment modes.
- For Amazon EKS and Kubernetes containers, see Example Kubernetes YAML files based on deployment modes.

Lambda Insights

CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. The solution collects, aggregates, and summarizes system-level metrics including CPU time, memory, disk, and network. It also collects, aggregates, and summarizes diagnostic information such as cold starts and Lambda worker shutdowns to help you isolate issues with your Lambda functions and resolve them quickly.

Lambda Insights uses a new CloudWatch Lambda extension, which is provided as a Lambda layer. When you install this extension on a Lambda function, it collects system-level metrics and emits a single performance log event for every invocation of that Lambda function. CloudWatch uses embedded metric formatting to extract metrics from the log events.

> **Note**
>
> The Lambda Insights agent is supported only on Lambda runtimes that use Amazon Linux 2 and Amazon Linux 2023.

For more information about Lambda extensions, see Using AWS Lambda extensions. For more information about embedded metric format, see Embedding metrics within logs.

You can use Lambda Insights with any Lambda function that uses a Lambda runtime that supports Lambda extensions. For a list of these runtimes, see Lambda Extensions API.

Pricing

For each Lambda function enabled for Lambda Insights, you only pay for what you use for metrics and logs. For a pricing example, see Amazon CloudWatch Pricing.

You are charged for the execution time consumed by the Lambda extension, in 1ms increments. For more information about Lambda pricing, see AWS Lambda Pricing.

Get started with Lambda Insights

To enable Lambda Insights on a Lambda function, you can use a one-click toggle in the Lambda console. Alternatively, you can use the AWS CLI, CloudFormation, the AWS Serverless Application Model CLI, or the AWS Cloud Development Kit (AWS CDK).

The following sections provide detailed instructions for completing these steps.

Topics

- Available versions of the Lambda Insights extension
- Using the console to enable Lambda Insights on an existing Lambda function
- Use the AWS CLI to enable Lambda Insights on an existing Lambda function
- Use the AWS SAM CLI to enable Lambda Insights on an existing Lambda function
- Use CloudFormation to enable Lambda Insights on an existing Lambda function
- Use the AWS CDK to enable Lambda Insights on an existing Lambda function
- Use Serverless Framework to enable Lambda Insights on an existing Lambda function
- Enable Lambda Insights on a Lambda container image deployment
- Update the Lambda Insights extension version on a function

Available versions of the Lambda Insights extension

This section lists the versions of the Lambda Insights extension, and the ARNs to use for these extensions in each AWS Region.

Topics

- x86-64 platforms
- ARM64 platforms

x86-64 platforms

This section lists the versions of the Lambda Insights extension for x86-64 platforms, and the ARNs to use for these extensions in each AWS Region.

> **Important**
>
> Lambda Insights extensions 1.0.317.0 and later don't support Amazon Linux 1.

1.0.776.0

Version 1.0.776.0 fixes a packaging issue introduced during a build system migration. In previous versions, the extension RPM bundled a wrapper script instead of the compiled binary. This version correctly packages the compiled binary. This resolves issues that prevented the extension from running.

This fix also ensures that the RPM correctly declares the required shared library dependencies. These dependencies (libc, libgcc_s, libpthread, libdl) are needed by the binary at runtime.

ARNs for version 1.0.776.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:66`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:71`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:66`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:66`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:60`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:60`

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:145023102084:layer:LambdaInsightsExtension:15
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:102
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:65
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:46
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:64
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:66
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:66
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:44
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:37
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:17
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:727646510379:layer:LambdaInsightsExtension:8

Region	ARN
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension:15
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:65
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:29
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:53
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:53
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:66
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:43
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:63
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:60
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:44
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:66
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:66

Region	ARN
Europe (Paris)	`arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:65`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:37`
Middle East (UAE)	`arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:39`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:56`
Mexico (Central)	`arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension:15`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:65`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:31`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:31`

1.0.660.0

Version 1.0.660.0 includes Lambda Insights extension support for Lambda functions running on Lambda Managed Instances. For functions running on Lambda Managed Instances, the extension will send an EMF log event once every minute, creating 12 CloudWatch Metrics at one-minute granularity.

ARNs for version 1.0.660.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:64
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:64
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:64
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:64
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:56
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:56
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:38
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:40
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:13
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:33
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:62
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:727646510379:layer:LambdaInsightsExtension:6

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:45
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:63
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:64
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:64
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:145023102084:layer:LambdaInsightsExtension:13
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension:13
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:91
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:63
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:25
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:51
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:51
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:64

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:64
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:64
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:56
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:63
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:40
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:61
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:39
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:33
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension:13
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:56
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:39
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:63

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:24
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:24

1.0.498.0

Version 1.0.498.0 includes bug fixes and performance improvements.

ARNs for version 1.0.498.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:60
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:60
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:60
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:60
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:52
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension:8
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:52

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:34
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:36
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:9
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:29
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:58
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:41
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:59
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:60
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:60
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:87
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:59
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:21

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:48
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:48
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:60
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:60
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:60
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:52
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:59
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:36
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:57
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:35
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:29
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension:8

Region	ARN
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:52
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:35
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:59
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:24
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:24
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:727646510379:layer:LambdaInsightsExtension:1
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:145023102084:layer:LambdaInsightsExtension:8

1.0.404.0

Version 1.0.404.0 reduces the binary size of the Lambda Insights extension from ~10MB to ~6MB, and subsequently reduces the zip size of the Lambda Insights extension layer from ~4MB to ~2.7MB. It also reduces the overall memory consumption of the agent from ~10MB to ~7MB.

ARNs for version 1.0.404.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:56

Region	ARN
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:56
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:56
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:56
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:47
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension:3
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:47
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:29
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:33
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:4
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:24
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:54
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:37

Region	ARN
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:55
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:56
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:56
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:83
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:55
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:16
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:46
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:46
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:56
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:56
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:56
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:47

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:55
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:31
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:53
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:30
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:23
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:47
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:30
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:55
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:24
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:24

1.0.391.0

> **Important**
>
> Version 1.0.391.0 changes how Lambda Insights collects and reports thread usage. In previous versions of the extension, the `threads_max` metric reported the maximum number of threads running in the process that has the process ID of 1. Starting with version 1.0.391.0, this metric reports the maximum number of threads that were running at any point of time during the function's execution, across all processes in the execution environment. This includes the Lambda function's processes, extensions' processes, system/runtime specific processes, and so on. This makes the `threads_max` metric more comprehensive while assessing the remaining threads left for your use. For more information about thread and process quotas, see Lambda quotas.

Version 1.0.391.0 also adds support for the Asia Pacific (Thailand) and Mexico (Central) Regions.

ARNs for version 1.0.391.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:55`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:55`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:55`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:55`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:46`

Region	ARN
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension:2
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:46
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:28
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:32
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:3
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:23
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:53
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:36
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:54
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:55
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:55
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:82

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:54
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:15
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:45
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:45
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:55
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:55
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:55
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:46
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:54
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:30
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:52
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:29

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:23
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension:2
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:46
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:29
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:54
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:23
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:23

1.0.333.0

Version 1.0.333.0 includes bug fixes.

ARNs for version 1.0.333.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:53
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:53

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:53
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:53
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:44
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:44
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:26
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:30
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension:1
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:21
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:34
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:52
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:53

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:53
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:80
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:52
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:13
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:53
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:53
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:53
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:44
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:52
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:28

Region	ARN
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:50`
Europe (Zurich)	`arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:27`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:21`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:44`
Middle East (UAE)	`arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:27`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:52`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:21`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:21`

1.0.317.0

Version 1.0.317.0 includes the removal of support for the Amazon Linux 1 platform, and bug fixes. It also includes support for AWS GovCloud (US) Regions.

ARNs for version 1.0.317.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:52
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:52
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:52
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:52
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:43
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:43
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:25
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:29
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:20
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:50
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:33
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:51

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:52
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:52
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:79
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:51
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:12
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:42
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:42
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:52
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:52
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:52
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:43
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:51

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:27
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:49
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:26
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:20
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:43
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:26
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:51
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension:19
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension:19

1.0.295.0

Version 1.0.295.0 includes dependency updates for all the compatible runtimes.

ARNs for version 1.0.295.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:51`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:51`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:51`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:51`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:42`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:42`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:24`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:28`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:19`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:49`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:32`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:50`

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:51
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:78
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:50
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:11
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:41
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:41
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:51
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:51
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:51
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:42
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:50

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:26
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:48
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:25
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:19
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:42
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:25
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:50

1.0.275.0

Version 1.0.275.0 includes important dependency updates for all the compatible runtimes.

ARNs for version 1.0.275.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:49
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:49

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:49
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:49
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:40
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:40
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:22
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:26
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:17
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:47
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:48
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:49
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:49

Region	ARN
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:76
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:48
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:9
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:39
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:39
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:49
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:49
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:49
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:40
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:48
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:24
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:46

Region	ARN
Europe (Zurich)	`arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:23`
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:17`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:40`
Middle East (UAE)	`arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:23`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:48`

1.0.273.0

Version 1.0.273.0 includes important bug fixes for all compatible runtimes, and adds support for Canada West (Calgary).

ARNs for version 1.0.273.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:45`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:45`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:45`

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:45
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:35
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:35
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:21
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:12
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:43
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:26
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:45
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:45
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:72

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension:4
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:36
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:36
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:45
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:45
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:45
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:35
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:44
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:19
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:42
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:17

Region	ARN
Israel (Tel Aviv)	`arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:12`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:35`
Middle East (UAE)	`arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:18`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:44`

1.0.229.0

Version 1.0.229.0 includes important bug fixes for all compatible runtimes, and adds support for the Israel (Tel Aviv) Region.

ARNs for version 1.0.229.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:38`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:38`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:38`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:38`

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:28
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:28
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:10
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:14
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension:5
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:36
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:19
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:37
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:38
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:38
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:60
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:37

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:29
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:29
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:38
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:38
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:38
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:28
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:37
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:12
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:35
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:11
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:28

Region	ARN
Middle East (UAE)	`arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:11`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:37`

1.0.178.0

Version 1.0.178.0 adds support for the following AWS Regions.

- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Europe (Spain)
- Europe (Zurich)
- Middle East (UAE)

ARNs for version 1.0.178.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:35`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:33`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:33`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:33`

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:25
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:25
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension:8
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension:11
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:31
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:32
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:33
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:33
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:50
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:32
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:26

Region	ARN
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:26
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:35
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:33
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:33
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:25
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:32
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension:10
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:30
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:25
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension:9
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:32

1.0.143.0

Version 1.0.143.0 includes bug fixes in compatibility with Python 3.7 and Go 1.x. The Python 3.6 Lambda runtime is being deprecated. For more information, see Lambda runtimes.

ARNs for version 1.0.143.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:21`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:21`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:20`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:21`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:13`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:13`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:21`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:2`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:20`

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:21
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:21
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:32
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:20
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:14
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:14
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:21
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:21
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:21
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:13
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:20

Region	ARN
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:13`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:20`

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about `tmp` directory storage quotas, see Lambda quotas.

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

ARNs for version 1.0.135.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:18`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:18`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:18`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:18`

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:11
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:11
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:18
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension:1
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:18
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:18
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:18
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:25
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:18
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:11
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:11
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:18

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:18
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:18
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:11
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:18
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:11
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:18

1.0.119.0

ARNs for version 1.0.119.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:16
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:16

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:16
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:16
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:9
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:9
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:16
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:23
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:16
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:9
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:9

Region	ARN
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:16`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:16`
Europe (London)	`arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:16`
Europe (Milan)	`arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:9`
Europe (Paris)	`arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:16`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:16`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:9`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:16`

1.0.98.0

This version removes unnecessary logging and also addresses an issue with the AWS Serverless Application Model CLI local invokes. For more information about this issue, see Adding LambdaInsightsExtension results in timeout with 'sam local invoke'.

ARNs for version 1.0.98.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:14
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:14
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:14
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension:8
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension:8
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:14
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:14
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:14
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:14
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:14
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:14

Region	ARN
China (Beijing)	`arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension:8`
China (Ningxia);	`arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension:8`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:14`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:14`
Europe (London)	`arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:14`
Europe (Milan)	`arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension:8`
Europe (Paris)	`arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:14`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:14`
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension:8`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:14`

1.0.89.0

This version corrects the performance event timestamp to always represent the start of the invocation of the function.

ARNs for version 1.0.89.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:12`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:12`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:12`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:12`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:12`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:12`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:12`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:12`
Asia Pacific (Tokyo)	`arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:12`
Canada (Central)	`arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:12`
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:12`

Region	ARN
Europe (Ireland)	`arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:12`
Europe (London)	`arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:12`
Europe (Paris)	`arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:12`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:12`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:12`

1.0.86.0

With version 1.0.54.0 of the extension, memory metrics were sometimes reported incorrectly and sometimes were higher than 100%. Version 1.0.86.0 corrects the memory measurement issue by using the same event data as Lambda platform metrics. This means that you may see a dramatic change in the recorded memory metric values. This is achieved by using the new Lambda Logs API. This provides a more accurate measurement of Lambda sandbox memory usage. However, something to be aware of is that the Lambda Logs API can't deliver platform report events if a function sandbox times out and is subsequently spun down. In this case. Lambda Insights is unable to record the invocation metrics. For more information about Lambda Logs API, see AWS Lambda Logs API.

New features in version 1.0.86.0

- Uses the Lambda Logs API to correct the memory metric. This solves the previous issue where memory statistics were greater than 100%.
- Introduces `Init Duration` as a new CloudWatch metric.
- Uses the invocation ARN to add a **version** dimension for aliases and invoked versions. If you are using Lambda aliases or versions to achieve incremental deployments (such as blue-green deployments), you can view your metrics based on the invoked alias. The **version** dimension is

not applied if the function doesn't use an alias or version. For more information, see Lambda function aliases.

- Adds a `billed_mb_ms field` to the performance events to display the cost per invoke. This does not consider any cost associated with provisioned concurrency.
- Adds `billed_duration` and `duration` fields to the performance events.

ARNs for version 1.0.86.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:11`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:11`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:11`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:11`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:11`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:11`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:11`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:11`

Region	ARN
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:11
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:11
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:11
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:11
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:11
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:11
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:11
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:11

1.0.54.0

Version 1.0.54.0 was the initial release of the Lambda Insights extension.

ARNs for version 1.0.54.0

The following table lists the ARNs to use for this version of the extension in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension:2

Region	ARN
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension:2
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:2
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension:2
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension:2
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension:2
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension:2
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension:2
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension:2

Region	ARN
Europe (Paris)	`arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension:2`
Europe (Stockholm)	`arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension:2`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension:2`

ARM64 platforms

This section lists the versions of the Lambda Insights extension for ARM64 platforms, and the ARNs to use for these extensions in each AWS Region.

> **Important**
>
> Lambda Insights extensions 1.0.317.0 and later don't support Amazon Linux 1.

1.0.660.0

Version 1.0.660.0 includes Lambda Insights extension support for Lambda functions running on Lambda Managed Instances. For functions running on Lambda Managed Instances, the extension will send an EMF log event once every minute, creating 12 CloudWatch Metrics at one-minute granularity.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:31`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:33`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:29`

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:31
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:30
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:30
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:30
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension-Arm64:7
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension-Arm64:7
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:33
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:727646510379:layer:LambdaInsightsExtension-Arm64:165
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:28
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:30
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:31

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:31
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:145023102084:layer:LambdaInsightsExtension-Arm64:7
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension-Arm64:7
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:42
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension-Arm64:7
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension-Arm64:4
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension-Arm64:4
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:31
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:31
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:31
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:30

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:29
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension-Arm64:7
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension-Arm64:7
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension-Arm64:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:30
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension-Arm64:7
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:6
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:6

1.0.498.0

Version 1.0.498.0 includes bug fixes and performance improvements.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:29
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:25
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:27
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:26
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:26
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:27
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:26
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:158895979263:layer:LambdaInsightsExtension-Arm64:3

Region	ARN
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:590183865173:layer:LambdaInsightsExtension-Arm64:3
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:761018874580:layer:LambdaInsightsExtension-Arm64:2
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:38
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:26
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:24
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:25
Canada West (Calgary)	arn:aws:lambda:ca-west-1:946466191631:layer:LambdaInsightsExtension-Arm64:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
Europe (Zurich)	arn:aws:lambda:eu-central-2:033019950311:layer:LambdaInsightsExtension-Arm64:3
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:27
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:27
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:25

Region	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:25
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:26
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:14
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:459530977127:layer:LambdaInsightsExtension-Arm64:3
Mexico (Central)	arn:aws:lambda:mx-central-1:879381266642:layer:LambdaInsightsExtension-Arm64:2
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:26
Middle East (UAE)	arn:aws:lambda:me-central-1:732604637566:layer:LambdaInsightsExtension-Arm64:3
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:25
China (Beijing)	arn:aws-cn:lambda:cn-north-1:488211338238:layer:LambdaInsightsExtension-Arm64:1
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:488211338238:layer:LambdaInsightsExtension-Arm64:1
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:6
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:6

1.0.404.0

Version 1.0.404.0 reduces the binary size of the Lambda Insights extension from ~9MB to ~5MB, and subsequently reduces the zip size of the Lambda Insights extension layer from ~3,7MB to ~2.5MB. It also reduces the overall memory consumption of the agent from ~11MB to ~7MB.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:23`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:25`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:21`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:23`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:21`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:21`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:9`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:21`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:25`
Asia Pacific (Osaka)	`arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:20`
Asia Pacific (Seoul)	`arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:22`

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:23
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:34
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:23
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:21
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:21
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:9
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:21

Region	ARN
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:21`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:6`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:6`

1.0.391.0

Important

Version 1.0.391.0 changes how Lambda Insights collects and reports thread usage. In previous versions of the extension, the `threads_max` metric reported the maximum number of threads running in the process that has the process ID of 1. Starting with version 1.0.391.0, this metric reports the maximum number of threads that were running at any point of time during the function's execution, across all processes in the execution environment. This includes the Lambda function's processes, extensions' processes, system/runtime specific processes, and so on. This makes the `threads_max` metric more comprehensive while assessing the remaining threads left for your use. For more information about thread and process quotas, see Lambda quotas.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:22`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:24`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:20`

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:22
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:8
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:24
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:19
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:21
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:22
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:22
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:33
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:20

Region	ARN
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:22
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:22
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:22
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:20
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:20
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:8
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:20
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:5
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:5

1.0.333.0

Version 1.0.333.0 includes bug fixes.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:22
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:6
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:22
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:20

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:31
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:18
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:6
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:18
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:3
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:3

1.0.317.0

Version 1.0.317.0 includes the removal of support for the Amazon Linux 1 platform, and bug fixes. It also includes support for AWS GovCloud (US) Regions.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:21
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:5
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:17

Region	ARN
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:21
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:30
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:19
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:17
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:17

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:5
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:17
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:17
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:122132214140:layer:LambdaInsightsExtension-Arm64:1
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:751350123760:layer:LambdaInsightsExtension-Arm64:1

1.0.295.0

Version 1.0.295.0 includes dependency updates for all compatible runtimes.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:20
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:4
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:20
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:15
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:17
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:29
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:18

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:18
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:16
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:4
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16

1.0.275.0

Version 1.0.275.0 includes bug fixes for all compatible runtimes and support for the Europe (Spain) and Asia Pacific (Hyderabad) Regions.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:18

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:891564319516:layer:LambdaInsightsExtension-Arm64:2
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:18
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:13
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:15
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:27

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:16
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:14
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:14
Europe (Spain)	arn:aws:lambda:eu-south-2:352183217350:layer:LambdaInsightsExtension-Arm64:2
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:14
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:14

1.0.273.0

Version 1.0.273.0 includes bug fixes for all compatible runtimes.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:14
US West (N. California)	arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:9
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
Africa (Cape Town)	arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:9
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:9
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:9
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:14
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:9
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:11
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:12

Region	ARN
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:23
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:12
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:12
Europe (Milan)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:9
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:10
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:10
Middle East (Bahrain)	arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:9
South America (São Paulo)	arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:10

1.0.229.0

Version 1.0.229.0 includes bug fixes for all compatible runtimes. It also adds support for the following Regions:

- US West (N. California)
- Africa (Cape Town)

- Asia Pacific (Hong Kong)
- Asia Pacific (Jakarta)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Canada (Central)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Middle East (Bahrain)
- South America (São Paulo)

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:5`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:7`
US West (N. California)	`arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:3`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:012438385374:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:519774774795:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:439286490199:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:7`

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:194566237122:layer:LambdaInsightsExtension-Arm64:2
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:4
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:11
Canada (Central)	arn:aws:lambda:ca-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:5
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:5
Europe (Spain)	arn:aws:lambda:eu-south-1:339249233099:layer:LambdaInsightsExtension-Arm64:2
Europe (Paris)	arn:aws:lambda:eu-west-3:580247275435:layer:LambdaInsightsExtension-Arm64:3
Europe (Stockholm)	arn:aws:lambda:eu-north-1:580247275435:layer:LambdaInsightsExtension-Arm64:3

Region	ARN
Middle East (Bahrain)	`arn:aws:lambda:me-south-1:285320876703:layer:LambdaInsightsExtension-Arm64:2`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:3`

1.0.135.0

Version 1.0.135.0 includes bug fixes for how Lambda Insights collects and reports disk and file descriptor usage. In previous versions of the extension, the `tmp_free` metric reported the maximum free space in the `/tmp` directory while a function runs. This version changes the metric to report the minimum value instead, making it more useful when assessing disk usage. For more information about `tmp` directory storage quotas, see Lambda quotas.

Version 1.0.135.0 also now reports file descriptor usage (`fd_use` and `fd_max`) as the maximum value across processes rather than reporting the operating system level.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:2`
US East (Ohio)	`arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:2`
US West (Oregon)	`arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Singapore)	`arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2`
Asia Pacific (Sydney)	`arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:2`

Region	ARN
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
Europe (Ireland)	arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:2
Europe (London)	arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:2

1.0.119.0

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
US East (Ohio)	arn:aws:lambda:us-east-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
US West (Oregon)	arn:aws:lambda:us-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:580247275435:layer:LambdaInsightsExtension-Arm64:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:580247275435:layer:LambdaInsightsExtension-Arm64:1

Region	ARN
Europe (Frankfurt)	`arn:aws:lambda:eu-central-1:580247275435:layer:LambdaInsightsExtension-Arm64:1`
Europe (Ireland)	`arn:aws:lambda:eu-west-1:580247275435:layer:LambdaInsightsExtension-Arm64:1`
Europe (London)	`arn:aws:lambda:eu-west-2:580247275435:layer:LambdaInsightsExtension-Arm64:1`

Using the console to enable Lambda Insights on an existing Lambda function

Use the following steps in the Lambda Console to enable Lambda Insights on an existing Lambda function.

To enable Lambda Insights on a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose the name of a function, and then select the **Configuration** tab on the following screen.
3. Under the **Configuration** tab, choose **Monitoring and operations tools** in the left navigation menu, and then choose **Edit**.

   You're directed to a screen where you can edit monitoring tools.
4. By **Lambda Insights enhanced monitoring**, choose **Edit**.
5. Under **CloudWatch Lambda Insights,** enable **Enhanced monitoring**, and then choose **Save**.

Use the AWS CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CLI to enable Lambda Insights on an existing Lambda function.

Step 1: Update function permissions

To update the function's permissions

- Attach the **CloudWatchLambdaInsightsExecutionRolePolicy** managed IAM policy to the function's execution role by entering the following command.

```
aws iam attach-role-policy \
--role-name function-execution-role \
--policy-arn "arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy"
```

Step 2: Install the Lambda extension

Install the Lambda extension by entering the following command. Replace the ARN value for the `layers` parameter with the ARN matching your Region and the extension version that you want to use. For more information, see Available versions of the Lambda Insights extension.

```
aws lambda update-function-configuration \
   --function-name function-name \
   --layers "arn:aws:lambda:us-west-1:580247275435:layer:LambdaInsightsExtension:14"
```

Step 3: Enable the CloudWatch Logs VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint.

If you need to do this step, enter the following command, replacing the placeholders with information for your VPC.

For more information, see Using CloudWatch Logs with Interface VPC Endpoints.

```
aws ec2 create-vpc-endpoint \
--vpc-id vpcId \
--vpc-endpoint-type Interface \
--service-name com.amazonaws.region.logs \
--subnet-id subnetId
--security-group-id securitygroupId
```

Use the AWS SAM CLI to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS SAM AWS CLI to enable Lambda Insights on an existing Lambda function.

If you don't already have the latest version of the AWS SAM CLI installed, you must first install or upgrade it. For more information, see Installing the AWS SAM CLI.

Step 1: Install the layer

To make the Lambda Insights extension available to all of your Lambda functions, add a `Layers` property to the `Globals` section of your SAM template with the ARN of the Lambda Insights layer. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

```
Globals:
  Function:
    Layers:
      - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

To enable this layer for only a single function, add the `Layers` property to the function as shown in this example.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

For each function, add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy.

AWS SAM doesn't support global policies, so you must enable those on each function individually, as shown in this example. For more information about globals, see Globals Section.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Policies:
        - CloudWatchLambdaInsightsExecutionRolePolicy
```

Invoking locally

The AWS SAM CLI supports Lambda extensions. However, every locally executed invocation resets the runtime environment. Lambda Insights data won't be available from local invocations because

the runtime is restarted without a shutdown event. For more information, see Release 1.6.0 - Add support for local testing of AWS Lambda extensions.

Troubleshooting

To troubleshoot your Lambda Insights installation, add the following environment variable to your Lambda function to enable debug logging.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Environment:
        Variables:
          LAMBDA_INSIGHTS_LOG_LEVEL: info
```

Use CloudFormation to enable Lambda Insights on an existing Lambda function

Follow these steps to use CloudFormation to enable Lambda Insights on an existing Lambda function.

Step 1: Install the layer

Add the Lambda Insights layer to the `Layers` property within the Lambda Insights layer ARN. The example below uses the layer for the initial release of Lambda Insights. For the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:580247275435:layer:LambdaInsightsExtension:14"
```

Step 2: Add the managed policy

Add the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to your function execution role.

```
Resources:
  MyFunctionExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy'
```

Step 3: (Optional) Add VPC endpoint

This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs virtual private cloud (VPC) endpoint. For more information, see Using CloudWatch Logs with Interface VPC Endpoints.

```
Resources:
  CloudWatchLogsVpcPrivateEndpoint:
    Type: AWS::EC2::VPCEndpoint
    Properties:
      PrivateDnsEnabled: 'true'
      VpcEndpointType: Interface
      VpcId: !Ref: VPC
      ServiceName: !Sub com.amazonaws.${AWS::Region}.logs
      SecurityGroupIds:
        - !Ref InterfaceVpcEndpointSecurityGroup
      SubnetIds:
        - !Ref PublicSubnet01
        - !Ref PublicSubnet02
        - !Ref PublicSubnet03
```

Use the AWS CDK to enable Lambda Insights on an existing Lambda function

Follow these steps to use the AWS CDK to enable Lambda Insights on an existing Lambda function. To use these steps, you must already be using the AWS CDK to manage your resources.

The commands in this section are in TypeScript.

First, update the function permissions.

```
executionRole.addManagedPolicy(
 ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaInsightsExecutionRolePolicy')
);
```

Next, install the extension on the Lambda function. Replace the ARN value for the `layerArn` parameter with the ARN matching your Region and the extension version that you want to use. For more information, see Available versions of the Lambda Insights extension.

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:580247275435:layer:LambdaInsightsExtension:14';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

If necessary, enable the virtual private cloud (VPC) endpoint for CloudWatch Logs. This step is necessary only for functions running in a private subnet with no internet access, and if you have not already configured a CloudWatch Logs VPC endpoint.

```
const cloudWatchLogsEndpoint = vpc.addInterfaceEndpoint('cwl-gateway', {
  service: InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
});

cloudWatchLogsEndpoint.connections.allowDefaultPortFromAnyIpv4();
```

Use Serverless Framework to enable Lambda Insights on an existing Lambda function

Follow these steps to use Serverless Framework to enable Lambda Insights on an existing Lambda function. For more information about Serverless Framework, see serverless.com.

This is done through a Lambda Insights plugin for Serverless. For more information, see serverless-plugin-lambda-insights.

If you don't already have the latest version of the Serverless command-line interface installed, you must first install or upgrade it. For more information, see Get started with Serverless Framework Open Source & AWS.

To use Serverless Framework to enable Lambda Insights on a Lambda function

1. Install the Serverless plugin for Lambda Insights by running the following command in your Serverless directory:

   ```
   npm install --save-dev serverless-plugin-lambda-insights
   ```

2. In your `serverless.yml` file, add the plugin in the `plugins` section as shown:

```
provider:
  name: aws
plugins:
  - serverless-plugin-lambda-insights
```

3. Enable Lambda Insights.

   - You can enable Lambda Insights individually per function by adding the following property to the serverless.yml file

```
functions:
  myLambdaFunction:
    handler: src/app/index.handler
    lambdaInsights: true #enables Lambda Insights for this function
```

   - You can enable Lambda Insights for all functions within the `serverless.yml` file by adding the following custom section:

```
custom:
  lambdaInsights:
    defaultLambdaInsights: true #enables Lambda Insights for all functions
```

4. Re-deploy the Serverless service by entering the following command:

```
serverless deploy
```

   This re-deploys all functions and enables Lambda Insights for those functions that you have specified. It enables Lambda Insights by adding the Lambda Insights layer and attaching the necessary permissions using the `arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy` IAM policy.

Enable Lambda Insights on a Lambda container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image, add lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image. The lines to add are different for x86-64 containers and ARM64 containers.

> **Note**
>
> The Lambda Insights agent is supported only on Lambda runtimes that use Amazon Linux 2 and Amazon Linux 2023.

Topics

- x86-64 container image deployment
- ARM64 container image deployment

x86-64 container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image running on an x86-64 container, add the following lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image.

```
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm
```

After you create your Lambda function, assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role, and Lambda Insights is enabled on the container image-based Lambda function.

> **Note**
>
> To use an older version of the Lambda Insights extension, replace the URL in the previous commands with this URL: `https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension.1.0.111.0.rpm`. For more information, see Available versions of the Lambda Insights extension.

To verify the signature of the Lambda Insights agent package on a Linux server

1. Enter the following command to download the public key.

```
shell$ wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
lambda-insights-extension.gpg
```

2. Enter the following command to import the public key into your keyring.

```
shell$ gpg --import lambda-insights-extension.gpg
```

The output will be similar to the following. Make a note of the `key` value, you will need it in the next step. In this example output, the key value is 848ABDC8.

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1  (RSA: 1)
```

3. Verify the fingerprint by entering the following command. Replace `key-value` with the value of the key from the preceding step.

```
shell$  gpg --fingerprint key-value
```

The fingerprint string in the output of this command should be `E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8`. If the string doesn't match, don't install the agent and contact AWS.

4. After you have verified the fingerprint, you can use it to verify the Lambda Insights agent package. Download the package signature file by entering the following command.

```
shell$  wget https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm.sig
```

5. Verify the signature by entering the following command.

```
shell$ gpg --verify lambda-insights-extension.rpm.sig lambda-insights-extension.rpm
```

The output should look like the following:

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

```
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

In the expected output, there might be a warning about a trusted signature. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

If the output contains `BAD signature`, check whether you performed the steps correctly. If you continue to get a `BAD signature` response, contact AWS and avoid using the downloaded file.

x86-64 Example

This section includes an example of enabling Lambda Insights on a container image-based Python Lambda function.

An example of enabling Lambda Insights on a Lambda container image

1. Create a Dockerfile that is similar to the following:

```
FROM public.ecr.aws/lambda/python:3.8

// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension.rpm && \
    rpm -U lambda-insights-extension.rpm && \
    rm -f lambda-insights-extension.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Create a Python file named `index.py` that is similar to the following:

```
def handler(event, context):
  return {
    'message': 'Hello World!'
  }
```

3. Put the Dockerfile and `index.py` in the same directory. Then, in that directory, run the following steps to build the Docker image and upload it to Amazon ECR.

```
// create an ECR repository
```

```
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. Use that Amazon ECR image that you just created to create the Lambda function.
5. Assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role.

ARM64 container image deployment

To enable Lambda Insights on a Lambda function that is deployed as a container image running on an AL2_aarch64 container (which uses ARM64 architecture), add the following lines in your Dockerfile. These lines install the Lambda Insights agent as an extension in your container image.

```
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/
amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm
```

After you create your Lambda function, assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role, and Lambda Insights is enabled on the container image-based Lambda function.

> **Note**
>
> To use an older version of the Lambda Insights extension, replace the URL in the previous commands with this URL: `https://lambda-insights-extension-arm64.s3-ap-northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.1.0.229.0.rpm`. For more information, see Available versions of the Lambda Insights extension.

To verify the signature of the Lambda Insights agent package on a Linux server

1. Enter the following command to download the public key.

```
shell$ wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/lambda-insights-extension.gpg
```

2. Enter the following command to import the public key into your keyring.

```
shell$ gpg --import lambda-insights-extension.gpg
```

   The output will be similar to the following. Make a note of the `key` value, you will need it in the next step. In this example output, the key value is 848ABDC8.

```
gpg: key 848ABDC8: public key "Amazon Lambda Insights Extension" imported
gpg: Total number processed: 1
gpg: imported: 1  (RSA: 1)
```

3. Verify the fingerprint by entering the following command. Replace `key-value` with the value of the key from the preceding step.

```
shell$  gpg --fingerprint key-value
```

   The fingerprint string in the output of this command should be `E0AF FA11 FFF3 5BD7 349E E222 479C 97A1 848A BDC8`. If the string doesn't match, don't install the agent and contact AWS.

4. After you have verified the fingerprint, you can use it to verify the Lambda Insights agent package. Download the package signature file by entering the following command.

```
shell$  wget https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm.sig
```

5. Verify the signature by entering the following command.

```
shell$ gpg --verify lambda-insights-extension-arm64.rpm.sig lambda-insights-
extension-arm64.rpm
```

   The output should look like the following:

```
gpg: Signature made Thu 08 Apr 2021 06:41:00 PM UTC using RSA key ID 848ABDC8
gpg: Good signature from "Amazon Lambda Insights Extension"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E0AF FA11 FFF3 5BD7 349E  E222 479C 97A1 848A BDC8
```

In the expected output, there might be a warning about a trusted signature. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

If the output contains `BAD signature`, check whether you performed the steps correctly. If you continue to get a `BAD signature` response, contact AWS and avoid using the downloaded file.

ARM64 Example

This section includes an example of enabling Lambda Insights on a container image-based Python Lambda function.

An example of enabling Lambda Insights on a Lambda container image

1. Create a Dockerfile that is similar to the following:

```
FROM public.ecr.aws/lambda/python:3.8
// extra lines to install the agent here
RUN curl -O https://lambda-insights-extension-arm64.s3-ap-
northeast-1.amazonaws.com/amazon_linux/lambda-insights-extension-arm64.rpm && \
    rpm -U lambda-insights-extension-arm64.rpm && \
    rm -f lambda-insights-extension-arm64.rpm

COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. Create a Python file named `index.py` that is similar to the following:

```
def handler(event, context):
  return {
    'message': 'Hello World!'
  }
```

3. Put the Dockerfile and `index.py` in the same directory. Then, in that directory, run the following steps to build the Docker image and upload it to Amazon ECR.

```
// create an ECR repository
aws ecr create-repository --repository-name test-repository
// build the docker image
docker build -t test-image .
// sign in to AWS
aws ecr get-login-password | docker login --username AWS --password-stdin
 "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com
// tag the image
docker tag test-image:latest "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/
test-repository:latest
// push the image to ECR
docker push "${ACCOUNT_ID}".dkr.ecr."${REGION}".amazonaws.com/test-
repository:latest
```

4. Use that Amazon ECR image that you just created to create the Lambda function.
5. Assign the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy to the function's execution role.

Update the Lambda Insights extension version on a function

As a best practice, we recommend that you keep your Lambda Insights extension updated to the latest version. The topics in this page explain how to do so.

> **Note**
>
> This page explains how to update the extension version used by a function that is already using Lambda Insights. For information about how to get started with Lambda Insights, see Get started with Lambda Insights.

Use the Lambda console to update the Lambda Insights extension version

Use the following steps to use the Lambda console to update the Lambda Insights extension version.

To update using the Lambda console

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose the name of your function.
3. In the **Layers** section, choose **Edit**.
4. In the list of layers, search for **LambdaInsightsExtension** and then change the layer version to the latest version listed in Available versions of the Lambda Insights extension.
5. Choose **Save**.

Use the AWS CLI to update the Lambda Insights extension version

To use the AWS CLI to update the Lambda Insights extension version, enter the following command. Replace the ARN value for the layers parameter with the ARN matching your Region and the extension version that you want to use. For information about the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

```
aws lambda update-function-configuration \
--function-name function-name \
--layers "arn:aws:lambda:us-west-1:111122223333:layer:LambdaInsightsExtension:53"
```

Use the AWS SAM CLI to update the Lambda Insights extension on one or more functions

To update the Lambda Insights Extension version for all of your Lambda functions, update the `Layers` property in the `Globals` section of your AWS Serverless Application Model (SAM) template with the ARN of the Lambda Insights layer. For information about the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

The following updates all of your Lambda functions.

```
Globals:
  Function:
    Layers:
       - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

The following updates just one function.

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

Use CloudFormation to update the Lambda Insights extension on one or more functions

To update the Lambda Insights Extension version by using CloudFormation, update the extension layer in the `Layers` property within the function's CloudFormation resource, as in the following example. For information about the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Layers:
        - !Sub "arn:aws:lambda:
${AWS::Region}:111122223333:layer:LambdaInsightsExtension:53"
```

Use the AWS CDK to update the Lambda Insights extension on one or more functions

You can update the extension version on the Lambda function by replacing the ARN value for the `layerArn` parameter with the ARN matching your Region and the extension version that you want to use. For information about the latest release version of the Lambda Insights extension layer, see Available versions of the Lambda Insights extension.

```
import lambda = require('@aws-cdk/aws-lambda');
const layerArn = 'arn:aws:lambda:us-
west-1:111122223333:layer:LambdaInsightsExtension:53';
const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
```

Use Serverless Framework to update the Lambda Insights extension on one or more functions

Follow these steps to use Serverless Framework to update the Lambda Insights extension version on an existing Lambda function. For more information about Serverless Framework, see the Serverless Framework documentation.

This method uses a Lambda Insights plugin for Serverless. For more information, see serverless-plugin-lambda-insights.

If you don't already have the latest version of the Serverless command-line interface installed, you must first install it or upgrade it. For more information, see Setting Up Serverless Framework With AWS.

To update using the Lambda console

1. Update Lambda Insights. If you haven't already done so, add a `custom` section at the end of the file and specify the Lambda Insights version inside a `lambdaInsightsVersion` property.

```
custom:
    lambdaInsights:
        lambdaInsightsVersion: 53 #specify the Layer Version
```

2. Re-deploy the Serverless service by entering the following command.

```
serverless deploy
```

Update the Lambda Insights extension version on a Lambda container image deployment

To update Lambda Insights on a Lambda container image, follow the steps in Enable Lambda Insights on a Lambda container image deployment to rebuild the image with the latest version of Lambda Insights. Then, use the AWS CLI to update the function code and provide a container image URI as the value for the `--image-uri` parameter.

Viewing your Lambda Insights metrics

After you have installed the Lambda Insights extension on a Lambda function that has been invoked, you can use the CloudWatch console to see your metrics. You can see a multi-function overview, or focus on a single function. You can also view metrics for Lambda functions running on Managed Instances.

For a list of Lambda Insights metrics, see Metrics collected by Lambda Insights.

To view the multi-function overview for your Lambda Insights metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Insights**, **Lambda Insights**.

3. In the **Lambda Insights view** dropdown at the top of the page, select **Functions**.
4. Choose **Multi-function**.

   The top part of the page displays graphs with aggregated metrics of all your Lambda functions in the Region that have Lambda Insights enabled. Lower on the page is a table that lists the functions.
5. To filter by function name to reduce the number of functions displayed, type part of the function name in the box near the top of the page.
6. To add this view to a dashboard as a widget, choose **Add to dashboard**.

To view metrics for a single function

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Insights**, **Lambda Insights**.
3. In the **Lambda Insights view** dropdown at the top of the page, select **Functions**.
4. Choose **Single function**.

   The top part of the page displays graphs with metrics for the selected function.
5. If you have X-Ray enabled, you can choose a single trace ID. This opens the X-Ray Trace Map page for that invocation, and from there you can zoom out to see the distributed trace and the other services involved in handling that specific transaction. For more information about the X-Ray Trace Map, see Using the X-Ray Trace Map.
6. To open CloudWatch Logs Insights and zoom in on a specific error, choose **View logs** by the table at the bottom of the page.
7. To add this view to a dashboard as a widget, choose **Add to dashboard**.

To view metrics for Lambda functions running on Managed Instances

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Insights**, **Lambda Insights**.
3. In the **Lambda Insights view** dropdown at the top of the page, select **Managed Instances Functions**.

   The page displays the Managed Instances view with metrics for your Lambda functions running on Managed Instances.

4. In the **Views** section, select a granularity level for viewing your metrics:
   - **Capacity Provider** – Displays aggregated metrics at the capacity provider level.
   - **Instance Type** – Displays aggregated metrics at the instance type level.
   - **Function** – Displays metrics at the individual function level. The top part of the page displays graphs with aggregated metrics of all your Lambda functions in the region that has Lambda Insights enabled. Lower on the page is a table that lists the functions. To filter by function name, type part of the function name in the box near the top of the page.
5. Use the **Filters** section to filter by capacity provider, instance type, or function.
6. To add this view to a dashboard as a widget, choose **Add to dashboard**.

Integration with Application Insights

Amazon CloudWatch Application Insights helps you monitor your applications and identifies and sets up key metrics, logs, and alarms across your application resources and technology stack. For more information, see Detect common application problems with CloudWatch Application Insights.

You can enable Application Insights to gather additional data from your Lambda functions. If you haven't done this already, you can enable it by choosing **Auto-configure Application Insights** in the **Application Insights** tab below the performance view in the Lambda Insights dashboard.

If you have already set up CloudWatch Application Insights to monitor your Lambda functions, the Application Insights dashboard appears below the Lambda Insights dashboard, in the **Application Insights** tab.

Metrics collected by Lambda Insights

Lambda Insights collects several metrics from the Lambda functions where it is installed. Some of these metrics are available as time series aggregated data in CloudWatch Metrics. Other metrics are not aggregated into time series data but can be found in the embedded metric format log entries by using CloudWatch Logs Insights.

Topics

- Lambda Functions
- Managed Instances Lambda Functions

Lambda Functions

The following metrics are available as time series aggregated data in CloudWatch Metrics in the `LambdaInsights` namespace for standard Lambda functions.

Metric name	Dimensions	Description
`cpu_total_time`	function_name function_name, version	Sum of `cpu_system_time` and `cpu_user_time`. Unit: Milliseconds
`init_duration`	function_name function_name, version	The amount of time spent in the `init` phase of the Lambda execution environment lifecycle. Unit: Milliseconds
`memory_utilization`	function_name function_name, version	The maximum memory measured as a percentage of the memory allocated to the function. Unit: Percent
`used_memory_max`	function_name function_name, version	The measured memory of the function's execution environment. Unit: Megabytes
`rx_bytes`	function_name function_name, version	The number of bytes received by the function.

Metric name	Dimensions	Description
		Unit: Bytes
`tmp_free`	function_name function_name, version	The amount of space available in the `/tmp` directory. Unit: Bytes
`tmp_used`	function_name function_name, version	The amount of space used in the `/tmp` directory. Unit: Bytes
`tx_bytes`	function_name function_name, version	The number of bytes sent by the function. Unit: Bytes
`total_memory`	function_name function_name, version	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes

Metric name	Dimensions	Description
total_network	function_name function_name, version	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes
used_memory_max	function_name function_name, version	The measured memory of the function sandbox. Unit: Megabytes

The following metrics can be found in the embedded metric format log entries by using CloudWatch Logs Insights. For more information about CloudWatch Logs Insights, see Analyzing Log Data with CloudWatch Logs Insights.

For more information about embedded metric format, see Embedding metrics within logs.

Metric name	Description	
agent_version	The current version of the Lambda Insights extension agent running on the Lambda function. Unit: String	
cpu_user_time	The amount of time the CPU spent executing user code.	

Metric name	Description
	Unit: Milliseconds
cpu_system_time	The amount of time the CPU spent executing kernel code. Unit: Milliseconds
cpu_total_time	Sum of cpu_system_time and cpu_user_time . Unit: Milliseconds
fd_use	File descriptors currently in use. Unit: Count
fd_max	Maximum file descriptors available for use. Unit: Count
version	The Lambda function's version for which the other metrics were collected. Unit: Count
agent_memory_max	Maximum memory consumed by the Lambda Insights extension agent. Unit: Bytes
agent_memory_avg	Average memory consumed by the Lambda Insights extension agent. Unit: Bytes

Metric name	Description
memory_utilization	The average memory measured as a percentage of the memory allocated to the execution environment. Unit: Percent
used_memory_max	The measured memory of the function's execution environment. Unit: Megabytes
rx_bytes	The number of bytes received by the function. Unit: Bytes
tx_bytes	The number of bytes sent by the function. Unit: Bytes
threads_max	The number of threads in use by the function process. As a function author, you don't control the initial number of threads created by the runtime. Unit: Count
tmp_used	The amount of space used in the /tmp directory. Unit: Bytes
tmp_max	The amount of space available in the /tmp directory. Unit: Bytes

Metric name	Description	
total_memory	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes	
total_network	Sum of rx_bytes and tx_bytes. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes	

Managed Instances Lambda Functions

The following metrics are available as time series aggregated data in CloudWatch Metrics in the LambdaInsights namespace for Lambda functions running on Managed Instances.

Metric name	Dimensions	Description
cpu_utilization	function_name function_name, version	The average CPU measured as a percentage of the vCPUs allocated to the execution environment. Unit: Percent
cpu_utilization_max	function_name function_name, version	The maximum CPU measured as a percentage of the vCPUs allocated to the execution environment. This is

Metric name	Dimensions	Description
		sampled at 1 second intervals. Unit: Percent
`cpu_total_time`	function_name function_name, version	Sum of `cpu_system_time` and `cpu_user_time`. Unit: Milliseconds
`memory_utilization`	function_name function_name, version	The average memory measured as a percentage of the memory allocated to the execution environment. Unit: Percent
`memory_utilization_max`	function_name function_name, version	The maximum memory measured as a percentage of the memory allocated to the execution environment. This is sampled at 50ms intervals. Unit: Percent
`used_memory_max`	function_name function_name, version	The measured memory of the function's execution environment. Unit: Megabytes

Metric name	Dimensions	Description
rx_bytes	function_name function_name, version	The number of bytes received by the function. Unit: Bytes
tmp_free	function_name function_name, version	The amount of space available in the /tmp directory. Unit: Bytes
tmp_used	function_name function_name, version	The amount of space used in the /tmp directory. Unit: Bytes
tx_bytes	function_name function_name, version	The number of bytes sent by the function. Unit: Bytes
total_memory	function_name function_name, version	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes

Metric name	Dimensions	Description
`total_network`	function_name function_name, version	Sum of `rx_bytes` and `tx_bytes`. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes
`used_memory_max`	function_name function_name, version	The measured memory of the function sandbox. Unit: Megabytes

The following metrics can be found in the embedded metric format log entries by using CloudWatch Logs Insights. For more information about CloudWatch Logs Insights, see Analyzing Log Data with CloudWatch Logs Insights.

For more information about embedded metric format, see Embedding metrics within logs.

Metric name	Description	
`lambda_mode`	Indicating if this log event is for a function running on Lambda managed instances or not. If this field is present in the log event, it can have only one value - `managed-instance` . The absence of this field indicates that the function was a regular Lambda function. Unit: String	

Metric name	Description
agent_version	The current version of the Lambda Insights extension agent running on the Lambda function. Unit: String
cpu_utilization	The average CPU measured as a percentage of the vCPUs allocated to the execution environment. Unit: Percent
cpu_utilization_max	The maximum CPU measured as a percentage of the vCPUs allocated to the execution environment. This is sampled at 1 second intervals. Unit: Percent
cpu_user_time	The amount of time the CPU spent executing user code. Unit: Milliseconds
cpu_system_time	The amount of time the CPU spent executing kernel code. Unit: Milliseconds
cpu_total_time	Sum of cpu_system_time and cpu_user_time . Unit: Milliseconds
fd_use	File descriptors currently in use. Unit: Count

Metric name	Description
fd_max	Maximum file descriptors available for use. Unit: Count
execution_environment_init	Boolean value indicating if a new execution environment was spun up for this function. Unit: Boolean
version	The Lambda function's version for which the other metrics were collected. Unit: Count
agent_memory_max	Maximum memory consumed by the Lambda Insights extension agent. Unit: Bytes
agent_memory_avg	Average memory consumed by the Lambda Insights extension agent. Unit: Bytes
memory_utilization	The average memory measured as a percentage of the memory allocated to the execution environment. Unit: Percent
memory_utilization_max	The maximum memory measured as a percentage of the memory allocated to the execution environment. This is sampled at 50ms intervals. Unit: Percent

Metric name	Description
used_memory_max	The measured memory of the function's execution environment. Unit: Megabytes
rx_bytes	The number of bytes received by the function. Unit: Bytes
tx_bytes	The number of bytes sent by the function. Unit: Bytes
threads_max	The number of threads in use by the function process. As a function author, you don't control the initial number of threads created by the runtime. Unit: Count
tmp_free	The amount of space available in the /tmp directory. Unit: Bytes
tmp_used	The amount of space used in the /tmp directory. Unit: Bytes
tmp_max	The amount of space available in the /tmp directory. Unit: Bytes

Metric name	Description
`total_memory`	The amount of memory allocated to your Lambda function. This is the same as your function's memory size. Unit: Megabytes
`total_network`	Sum of `rx_bytes` and `tx_bytes`. Even for functions that don't perform I/O tasks, this value is usually greater than zero because of network calls made by the Lambda runtime. Unit: Bytes

Troubleshooting and known issues for CloudWatch Lambda Insights

The first step to troubleshooting any issues is to enable debug logging on the Lambda Insights extension. To do this, set the following environment variable on your Lambda function: `LAMBDA_INSIGHTS_LOG_LEVEL=info`. For more information, see Using AWS Lambda environment variables.

The extension emits logs into the same log group as your function (`/aws/lambda/`*`function-name`*). Review those logs to see if the error might be related to a setup issue.

I don't see any metrics from Lambda Insights

If you don't see Lambda Insights metrics that you expect to see, check the following possibilities:

- **The metrics might just be delayed**— If the function has not yet been invoked or the data has not been flushed yet, you won't see the metrics in CloudWatch. For more information, see **Known Issues** later in this section.
- **Confirm that the Lambda function has the correct permissions**—Make sure that the **CloudWatchLambdaInsightsExecutionRolePolicy** IAM policy is assigned to the function's execution role.
- **Check the Lambda runtime**—Lambda Insights supports only certain Lambda runtimes. For a list of supported runtimes, see Lambda Insights.

For example, to use Lambda Insights on Java 8, you must use the `java8.al2` runtime, not the `java8` runtime.

- **Check network access**—The Lambda function might be on a VPC private subnet with no internet access and you don't have a VPC endpoint configured for CloudWatch Logs. To help debug this issue, you can set the environment variable `LAMBDA_INSIGHTS_LOG_LEVEL=info`.

Known issues

Data delay can be as high as 20 minutes. When a function handler completes, Lambda freezes the sandbox, which also freezes the Lambda Insights extension. While the function is running, we use an adaptive batching strategy based on the function TPS to output data. However, if the function stops being invoked for an extended period and there is still event data in the buffer, this data can be delayed until Lambda shuts down the idle sandbox. When Lambda shuts down the sandbox, we flush the buffered data.

Example telemetry event in CloudWatch Lambda Insights

Each invocation of a Lambda function that has Lambda Insights enabled writes a single log event to the `/aws/lambda-insights` log group. Each log event contains metrics in embedded metric format. For more information about embedded metric format, see Embedding metrics within logs.

To analyze these log events, you can use the following methods:

- The Lambda Insights section of the CloudWatch console, as explained in Viewing your Lambda Insights metrics.
- Log event queries using CloudWatch Logs Insights. For more information, see Analyzing Log Data with CloudWatch Logs Insights.
- Metrics collected in the `LambdaInsights` namespace, which you graph by using CloudWatch metrics.

The following is an example of a Lambda Insights log event with embedded metric format.

```
{
    "_aws": {
        "Timestamp": 1605034324256,
        "CloudWatchMetrics": [
            {
```

```
                "Namespace": "LambdaInsights",
                "Dimensions": [
                    [ "function_name" ],
                    [ "function_name", "version" ]
                ],
                "Metrics": [
                    { "Name": "memory_utilization", "Unit": "Percent" },
                    { "Name": "total_memory", "Unit": "Megabytes" },
                    { "Name": "used_memory_max", "Unit": "Megabytes" },
                    { "Name": "cpu_total_time", "Unit": "Milliseconds" },
                    { "Name": "tx_bytes", "Unit": "Bytes" },
                    { "Name": "rx_bytes", "Unit": "Bytes" },
                    { "Name": "total_network", "Unit": "Bytes" },
                    { "Name": "init_duration", "Unit": "Milliseconds" }
                ]
            }
        ],
        "LambdaInsights": {
            "ShareTelemetry": true
        }
    },
    "event_type": "performance",
    "function_name": "cpu-intensive",
    "version": "Blue",
    "request_id": "12345678-8bcc-42f7-b1de-123456789012",
    "trace_id": "1-5faae118-12345678901234567890",
    "duration": 45191,
    "billed_duration": 45200,
    "billed_mb_ms": 11571200,
    "cold_start": true,
    "init_duration": 130,
    "tmp_free": 538329088,
    "tmp_max": 551346176,
    "threads_max": 11,
    "used_memory_max": 63,
    "total_memory": 256,
    "memory_utilization": 24,
    "cpu_user_time": 6640,
    "cpu_system_time": 50,
    "cpu_total_time": 6690,
    "fd_use": 416,
    "fd_max": 32642,
    "tx_bytes": 4434,
    "rx_bytes": 6911,
```

```
    "timeout": true,
    "shutdown_reason": "Timeout",
    "total_network": 11345,
    "agent_version": "1.0.72.0",
    "agent_memory_avg": 10,
    "agent_memory_max": 10
}
```

The following is an example of a Lambda Insights log event for a Lambda function running on Lambda Managed Instances.

```
{
    "total_network": 16443,
    "tmp_free": 531492864,
    "total_memory": 2048,
    "fd_use": 85,
    "tmp_used": 11984896,
    "execution_environment_init": false,
    "version": "3",
    "event_type": "performance",
    "agent_memory_max": 6,
    "fd_max": 1024,
    "function_name": "cpu-intensive",
    "tx_bytes": 8404,
    "memory_utilization": 3,
    "used_memory_max": 73,
    "memory_utilization_max": 3,
    "cpu_system_time": 541,
    "threads_max": 49,
    "tmp_max": 543477760,
    "cpu_utilization_max": 2,
    "agent_memory_avg": 6,
    "cpu_total_time": 815,
    "rx_bytes": 8039,
    "lambda_mode": "managed-instance",
    "agent_version": "1.0.660.0",
    "_aws": {
        "CloudWatchMetrics": [
            {
                "Namespace": "LambdaInsights",
                "Dimensions": [
                    [
                        "function_name"
```

```
                    ],
                    [
                        "function_name",
                        "version"
                    ]
                ],
                "Metrics": [
                    {
                        "Name": "cpu_total_time",
                        "Unit": "Milliseconds"
                    },
                    {
                        "Name": "cpu_utilization",
                        "Unit": "Percent"
                    },
                    {
                        "Name": "cpu_utilization_max",
                        "Unit": "Percent"
                    },
                    {
                        "Name": "tx_bytes",
                        "Unit": "Bytes"
                    },
                    {
                        "Name": "rx_bytes",
                        "Unit": "Bytes"
                    },
                    {
                        "Name": "total_network",
                        "Unit": "Bytes"
                    },
                    {
                        "Name": "used_memory_max",
                        "Unit": "Megabytes"
                    },
                    {
                        "Name": "memory_utilization",
                        "Unit": "Percent"
                    },
                    {
                        "Name": "memory_utilization_max",
                        "Unit": "Percent"
                    },
                    {
```

```
                        "Name": "total_memory",
                        "Unit": "Megabytes"
                    },
                    {
                        "Name": "tmp_used",
                        "Unit": "Bytes"
                    },
                    {
                        "Name": "tmp_free",
                        "Unit": "Bytes"
                    }
                ]
            }
        ],
        "Timestamp": 1764164871353,
        "LambdaInsights": {
            "ShareTelemetry": true
        }
    },
    "cpu_utilization": 1,
    "cpu_user_time": 273
}
```

CloudWatch Database Insights

> **Note**
>
> AWS announced that the end-of-life date for Performance Insights was July 31, 2026, and has migrated Performance Insights users to Database Insights. Standard mode supports flexible retention of 1–24 months at the same price as Performance Insights. Advanced mode supports flexible retention of 1–24 months, at no extra cost.

Use CloudWatch Database Insights to monitor and troubleshoot Amazon Aurora MySQL, Amazon Aurora PostgreSQL, Amazon Aurora PostgreSQL Limitless, Amazon Aurora DSQL, Amazon RDS for SQL Server, RDS for MySQL, RDS for PostgreSQL, RDS for Oracle, RDS for MariaDB, and self-managed PostgreSQL databases at scale.

With Database Insights, you can monitor your database fleet with pre-built, opinionated dashboards. To help you analyze the performance of your fleet, the Database Insights dashboards display curated metrics and visualizations, and you can customize these dashboards. By presenting metrics in a single dashboard for all databases in your fleet, Database Insights allows you to monitor your databases simultaneously.

For example, you can use Database Insights to find a database that is performing poorly within a fleet of hundreds of database instances. You can then choose that instance and use Database Insights to troubleshoot issues.

For information about engine, AWS Region, and instance class support, see Aurora DB engine, Region, and instance class support for Database Insights and Amazon RDS DB engine, Region, and instance class support for Database Insights.

For information about monitoring Amazon Aurora DSQL, see Monitoring and logging for Aurora DSQL.

Database Insights also supports monitoring self-managed databases running on Amazon Elastic Compute Cloud. The supported engine for self-managed databases is PostgreSQL, see Monitoring Self-Managed Databases.

Database Insights supports monitoring workloads across multiple accounts and regions. To learn more about the cross-account cross-region monitoring feauture of Database Insights see Set up cross-account cross-region monitoring for CloudWatch Database Insights

To get started with Database Insights, see the following topics.

Topics

- Get started with CloudWatch Database Insights
- Viewing the Fleet Health Dashboard for CloudWatch Database Insights
- Viewing the Database Instance Dashboard for CloudWatch Database Insights
- Troubleshooting for CloudWatch Database Insights

Modes for Database Insights

Database Insights modes apply to Amazon Relational Database Service, Amazon Aurora provisioned, Amazon Aurora Serverless, and Amazon Aurora Limitless databases.

Database Insights has an Advanced mode and a Standard mode. Standard mode is the default for Database Insights, and you can turn on the Advanced mode for your database.

The following table shows which features CloudWatch supports for the Advanced mode and Standard mode of Database Insights.

Feature	Standard mode	Advanced mode
Analyze the top contributors to DB Load by dimension	Supported	Supported
Retention of detailed database and per-query metrics	Free up to 7 days, and paid for 1–24 months	1–24 months retention period included at no additional cost
Query, graph, and set alarms on database metrics with up to 7 days of retention	Supported	Supported
Define fine-grained access control policies to restrict access to potentially sensitive dimensions such as SQL text	Supported	Supported

Feature	Standard mode	Advanced mode
Analyze operating system processes happening in your databases with detailed metrics per running process Amazon RDS Enhanced Monitoring is required for this feature to work.	Not supported	Supported
Define and save fleet-wide monitoring views to assess database health at scale	Not supported	Supported
Analyze SQL locks with 15 months of retention and a guided UX	Not supported	Supported only for Aurora PostgreSQL and RDS for PostgreSQL
Analyze SQL execution plans with 15 months of retention and guided UX	Not supported	Supported only for Aurora PostgreSQL, RDS for Oracle, and RDS for SQL Server
Visualize per-query statistics	Not supported	Supported
Analyze slow SQL queries Export of database logs to CloudWatch Logs is required for this feature to work.	Not supported	Supported
View calling services with CloudWatch Application Signals	Not supported	Supported
View a consolidated dashboard for all database telemetry, including metrics, logs, events, and applications Export of database logs to CloudWatch Logs is required to view database logs in the Database Insights console.	Not supported	Supported

Feature	Standard mode	Advanced mode
Import Performance Insights counter metrics into CloudWatch automatically	Not supported	Supported
View Amazon RDS events in CloudWatch	Not supported	Supported
Analyze database performance for a time period of your choice with on-demand analysis	Not supported	Supported

Note

The Standard mode and Advanced mode apply to Amazon RDS and Amazon Aurora databases. Self-managed databases do not use modes. They provide a single monitoring experience with 15 months of metric retention, configured through the CloudWatch agent. For more information, see Monitoring Self-Managed Databases.

Note

Database Insights feature availability differs in different AWS Regions, because not all Advanced Mode features are available in all Regions.

Data retention

If you enable Database Insights Standard mode, Database Insights includes up to 7 days of detailed database and per-query metrics retention at no additional cost. For retention beyond 7 days, you can configure retention from 1 to 24 months. For pricing information, see Database Insights pricing.

Database Insights Advanced mode supports retention of detailed database and per-query metrics for 1–24 months, at no additional cost.

For information about detailed database and per-query metrics, see Amazon CloudWatch metrics for Amazon Aurora.

For information about the CloudWatch metrics collected by Database Insights, see the following topics.

- Amazon CloudWatch metrics for Amazon Aurora in the *Amazon Aurora User Guide*
- Amazon CloudWatch metrics for Amazon Relational Database Service in the *Amazon RDS User Guide*
- Amazon CloudWatch metrics for Amazon RDS Performance Insights in the *Amazon Aurora User Guide*
- Amazon CloudWatch metrics for Amazon RDS Performance Insights in the *Amazon Aurora User Guide*

Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Get started with CloudWatch Database Insights

The Standard mode of Database Insights is enabled by default for your Amazon RDS and Aurora databases. To get started with the Advanced mode of Database Insights, you can create a new database or modify a database.

For information about enabling the Advanced mode or the Standard mode of Database Insights for an Amazon RDS database, see the following topics.

- Turning on the Advanced mode of Database Insights for Amazon RDS in the *Amazon RDS User Guide*
- Turning on the Standard mode of Database Insights for Amazon RDS in the *Amazon RDS User Guide*
- Turning CloudWatch Database Insights on or off when creating a DB instance or Multi-AZ DB cluster for Amazon RDS in the *Amazon RDS User Guide*

For information about enabling the Advanced mode or the Standard mode of Database Insights for an Amazon Aurora database, see the following topics.

- Turning on the Advanced mode of Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*

- Turning on the Standard mode of Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*

For information about enabling the Advanced mode or the Standard mode of Database Insights for an Aurora PostgreSQL Limitless Database, see the following topics.

- Turning on the Advanced mode of Database Insights for Aurora PostgreSQL Limitless Database in the *Amazon Aurora User Guide*
- Turning on the Standard mode of Database Insights for Aurora PostgreSQL Limitless Database in the *Amazon Aurora User Guide*

For information about setting up monitoring for a self-managed database, see the following topic.

- Monitoring Self-Managed PostgreSQL

Required permissions for Database Insights

Certain IAM permissions are required to use Database Insights. Database Insights requires permissions for CloudWatch, CloudWatch Logs, Amazon RDS, and Amazon RDS Performance Insights. You might not need to provide these permissions to your user or role if you have broader permissions.

The following CloudWatch permissions are required to use Database Insights.

- `cloudwatch:BatchGetServiceLevelIndicatorReport`
- `cloudwatch:DescribeAlarms`
- `cloudwatch:GetDashboard`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `cloudwatch:PutDashboard`

The following CloudWatch Logs permissions are required to use Database Insights.

- `logs:DescribeLogGroups`
- `logs:GetQueryResults`
- `logs:StartQuery`

- `logs:StopQuery`

The following Amazon RDS permissions are required to use Database Insights.

- `rds:DescribeDBClusters`
- `rds:DescribeDBInstances`
- `rds:DescribeEvents`
- `rds:DescribeDBShardGroups` (if you are monitoring Aurora PostgreSQL Limitless Databases

The following Performance Insights permissions are required to use Database Insights.

- `pi:ListAvailableResourceMetrics`
- `pi:ListAvailableResourceDimensions`
- `pi:DescribeDimensionKeys`
- `pi:GetDimensionKeyDetails`
- `pi:GetResourceMetrics`
- `pi:ListPerformanceAnalysisReports`
- `pi:GetResourceMetadata`
- `pi:GetPerformanceAnalysisReport`
- `pi:CreatePerformanceAnalysisReport`
- `pi:DeletePerformanceAnalysisReport`
- `pi:ListTagsForResource`
- `pi:TagResource`
- `pi:UntagResource`

The following sample policy contains the permissions required for full access to Database Insights.

Sample policy for full access

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
```

```
    "Effect" : "Allow",
      "Action" : [
        "cloudwatch:BatchGetServiceLevelIndicatorReport",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutDashboard"
      ],
      "Resource" : "*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "logs:DescribeLogGroups",
        "logs:GetQueryResults",
        "logs:StartQuery",
        "logs:StopQuery"
      ],
      "Resource" : "*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "pi:DescribeDimensionKeys",
        "pi:GetDimensionKeyDetails",
        "pi:GetResourceMetadata",
        "pi:GetResourceMetrics",
        "pi:ListAvailableResourceDimensions",
        "pi:ListAvailableResourceMetrics",
        "pi:CreatePerformanceAnalysisReport",
        "pi:GetPerformanceAnalysisReport",
        "pi:ListPerformanceAnalysisReports",
        "pi:DeletePerformanceAnalysisReport",
        "pi:TagResource",
        "pi:UntagResource",
        "pi:ListTagsForResource"
      ],
      "Resource" : "arn:aws:pi:*:*:*/rds/*"
    },
    {
    "Effect" : "Allow",
      "Action" : [
        "rds:DescribeDBInstances",
```

```
        "rds:DescribeDBClusters",
        "rds:DescribeEvents"
      ],
      "Resource" : "*"
    }
  ]
}
```

Monitoring Self-Managed Databases

CloudWatch Database Insights supports monitoring self-managed databases, so you can bring databases that you run yourself into the same console you use for your Amazon RDS and Amazon Aurora databases. You install the CloudWatch agent on the database host, and within minutes the database appears in the Database Insights fleet view with live performance data, including database load (measured as average active sessions), wait event analysis, and query-level statistics.

Database Insights currently supports PostgreSQL for self-managed databases. The database can run on an Amazon EC2 instance.

By monitoring self-managed databases alongside your Amazon RDS and Amazon Aurora databases, you can use a single tool across heterogeneous fleets of managed and self-managed databases, apply consistent diagnostics, and troubleshoot performance issues from a single console.

Self-managed databases do not use the Standard mode and Advanced mode of Database Insights. Database Insights retains 15 months of the metrics collected for your self-managed databases. Logs collected by the agent are stored in CloudWatch Logs and follow the retention setting on the log group they are delivered to.

Standard pricing for CloudWatch OpenTelemetry metrics and CloudWatch Logs applies. For more information, see Amazon CloudWatch Pricing.

For information about setting up monitoring for a self-managed database, see the following topics.

Topics

- Monitoring Self-Managed PostgreSQL

Monitoring Self-Managed PostgreSQL

The following sections describe how to set up CloudWatch Database Insights monitoring for a self-managed PostgreSQL database. You prepare the database, install the CloudWatch agent on the database host, configure the agent to collect from the database, and verify that telemetry appears in the console.

Prerequisites

Before you set up monitoring for a self-managed PostgreSQL database, confirm the following.

- The database is running PostgreSQL version 14 or later.
- The CloudWatch agent runs on the same host as the PostgreSQL instance. Monitoring a remote database is not supported.
- You can create a dedicated, least-privilege database user for the agent to use for monitoring.
- The Amazon EC2 instance has an IAM role that can publish metrics and logs to CloudWatch. For more information, see Install the CloudWatch agent.

Prepare your PostgreSQL database

Self-managed monitoring reads performance data from standard PostgreSQL views, including `pg_stat_activity` and `pg_stat_statements`. Complete the following database-side configuration before you configure the agent.

To prepare a PostgreSQL database for Database Insights

1. In `postgresql.conf`, enable the `pg_stat_statements` extension and the server settings that Database Insights depends on. The extension must be loaded at startup through `shared_preload_libraries`, and the server logging settings populate the log files that the agent collects.

```
# Required
shared_preload_libraries = 'pg_stat_statements'
track_activities = on
track_activity_query_size = 4096
password_encryption = 'scram-sha-256'

# pg_stat_statements tuning
pg_stat_statements.max = 10000
```

```
pg_stat_statements.track = all
pg_stat_statements.track_planning = on

# Server logging (required for the slow query log)
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_rotation_age = 1d
log_rotation_size = 0
log_truncate_on_rotation = on
log_min_duration_statement = 500
log_line_prefix = '%m [%p] %q%u@%d '

# Recommended (PostgreSQL 14 and later)
compute_query_id = on
```

The day-of-week `log_filename` pattern (`postgresql-%a.log`), combined with daily rotation and `log_truncate_on_rotation`, caps the log directory at roughly seven files and reuses them each week. This keeps the query log from growing without bound and filling the volume on a long-running host. The `postgresql-*.log` glob that you configure for the agent later still matches these file names.

2. Restart PostgreSQL. A restart, rather than a reload, is required because `shared_preload_libraries` is read only at startup.

```
sudo systemctl restart postgresql
```

3. Create the `pg_stat_statements` extension in the `postgres` database. The agent connects to the `postgres` database and reads cluster-wide statistics from it, so the extension must exist there.

```
\c postgres
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
```

4. Create a dedicated monitoring user and grant it the `pg_monitor` role. We don't recommend using an admin account. The `pg_monitor` role grants read access to `pg_stat_activity`, `pg_stat_statements`, and other monitoring views without superuser privileges.

```
CREATE ROLE cw_monitor WITH LOGIN PASSWORD 'your-password';
GRANT pg_monitor TO cw_monitor;
```

5. To capture query execution plans, grant the monitoring user read access to the schemas that hold the tables your queries run against. Database Insights captures plans by running `EXPLAIN` as the monitoring user, and `EXPLAIN` requires access to the referenced tables. Run the grants in each database whose queries you want plans for, and repeat for any additional schemas.

```
GRANT USAGE ON SCHEMA public TO cw_monitor;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO cw_monitor;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO cw_monitor;
```

> **Note**
>
> The `ALTER DEFAULT PRIVILEGES` statement grants `SELECT` only on tables created later by the same role that runs the statement. If tables are created by a different role, run `ALTER DEFAULT PRIVILEGES FOR ROLE` *`owner`* for that role as well.

6. In `pg_hba.conf`, allow the monitoring user to connect over `localhost`. Add lines for both the IPv4 and IPv6 loopback addresses, because `localhost` can resolve to either one.

```
# TYPE  DATABASE  USER         ADDRESS         METHOD
host    all       cw_monitor   127.0.0.1/32    scram-sha-256
host    all       cw_monitor   ::1/128         scram-sha-256
```

   Then reload PostgreSQL to apply the change.

```
sudo systemctl reload postgresql
```

7. Verify that the monitoring user can connect over `localhost` and read the monitoring views.

```
psql -h localhost -U cw_monitor -d postgres -c "SELECT count(*) FROM
 pg_stat_activity;"
psql -h localhost -U cw_monitor -d postgres -c "SELECT count(*) FROM
 pg_stat_statements;"
```

Install the CloudWatch agent

Self-managed monitoring uses the CloudWatch agent to collect telemetry from your database. Install the agent on the same host as the PostgreSQL instance. There are several methods for installing the agent, so choose the one that best supports your operational practices. For full

instructions on downloading, installing, and setting up permissions for the agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The IAM role that the Amazon EC2 instance uses must have the `CloudWatchAgentServerPolicy` AWS managed policy attached. This policy grants the permissions the agent needs to publish metrics and logs to CloudWatch.

Configure the CloudWatch agent

After the agent is installed, add a `database_insights` section to the agent configuration that defines the PostgreSQL instance to monitor.

To configure the agent for a PostgreSQL database

1. Create a password file in PostgreSQL libpq pgpass format so that the monitoring user's password is kept out of the main agent configuration. Create `/opt/aws/amazon-cloudwatch-agent/etc/pgpass` with a single line.

   ```
   # hostname:port:database:username:password
   localhost:5432:*:cw_monitor:your-password
   ```

   Restrict the file so that only the agent user can read it.

   ```
   sudo chmod 600 /opt/aws/amazon-cloudwatch-agent/etc/pgpass
   sudo chown cwagent:cwagent /opt/aws/amazon-cloudwatch-agent/etc/pgpass
   ```

2. Add the `database_insights` section to the agent configuration file at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`.

   ```
   {
     "agent": {
       "region": "your-region"
     },
     "opentelemetry": {
       "collect": {
         "database_insights": {
           "postgresql": [
             {
               "endpoint": "localhost:5432",
               "instance_name": "your-instance-name",
               "username": "cw_monitor",
   ```

```
            "password_file": "/opt/aws/amazon-cloudwatch-agent/etc/pgpass",
            "logs": {
              "file_path": "/var/lib/pgsql/data/log/postgresql-*.log"
            }
          }
        ]
      }
    }
  }
}
```

The following table describes the key parameters.

Parameter	Description
`region`	The AWS Region to publish telemetry to.
`endpoint`	The host and port the agent connects to. Use `localhost` and the local PostgreSQL port.
`instance_name`	The display name for this instance in the Database Insights fleet view.
`username`	The monitoring user that you created.
`password_file`	The path to the libpq pgpass file that holds the monitoring user's password. The agent matches the `endpoint` host and port and the `username` against the entries in this file.
`logs.file_path`	A glob pattern for the PostgreSQL server log files to collect. Set this to match the `log_directory` configured on the database. The base directory is distribution-dependent: on RHEL and Amazon Linux the data directory is typically `/var/lib/pgsql/data/log/`, while on Debian and Ubuntu it is typically `/var/lib/postgresql/`*`version`*`/main/log/`. The example above uses the RHEL and Amazon Linux path; adjust it for your distribution.

3. Start the agent with the configuration. On an Amazon EC2 instance, use `-m ec2`.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
    -a fetch-config \
    -m ec2 \
    -s \
    -c file:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
```

4. Confirm that the agent is running.

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a status
```

Verify monitoring in the console

After the agent starts, open the CloudWatch console and choose **Database Insights**. Within a few minutes, your self-managed PostgreSQL instance appears in the fleet view with a non-zero **DB Load** metric. Choose the instance to open its dashboard, and confirm that the following panels show data.

- **DB Load** – active sessions broken down by wait event type, collected from `pg_stat_activity`.
- **Top SQL** – query statistics from `pg_stat_statements`.
- **Wait Events** – a breakdown by wait event type.
- **Host Metrics** – CPU, memory, and disk I/O collected by the agent.

If the fleet view shows the instance but the detailed panels are empty, allow a few more minutes for `pg_stat_statements` data to accumulate.

Set up cross-account cross-region monitoring for CloudWatch Database Insights

CloudWatch Database Insights supports cross-account and cross-region database fleet monitoring, enabling centralized observability across your entire database AWS infrastructure. It allows you to monitor, troubleshoot, and optimize databases spanning multiple AWS accounts and regions from a single unified console experience.

Topics

- Prerequisites
- Set-up your monitoring and source accounts for Database Insights cross-account access

- Set-up Database Insights for cross-account cross-region console
- Using Database Insights cross-account cross-region dashboards

Prerequisites

- Database Insights cross-account cross-region requires both **CloudWatch cross-account observability** and the **cross-account cross-region CloudWatch console** to be setup first. See instructions below on how to enable both.
- If you have already enabled them, you might still need to configure additional data sharing for Database Insights to work across accounts and regions in your environment. See instructions below and make sure you have selected the right data sharing configuration.

Set-up your monitoring and source accounts for Database Insights cross-account access

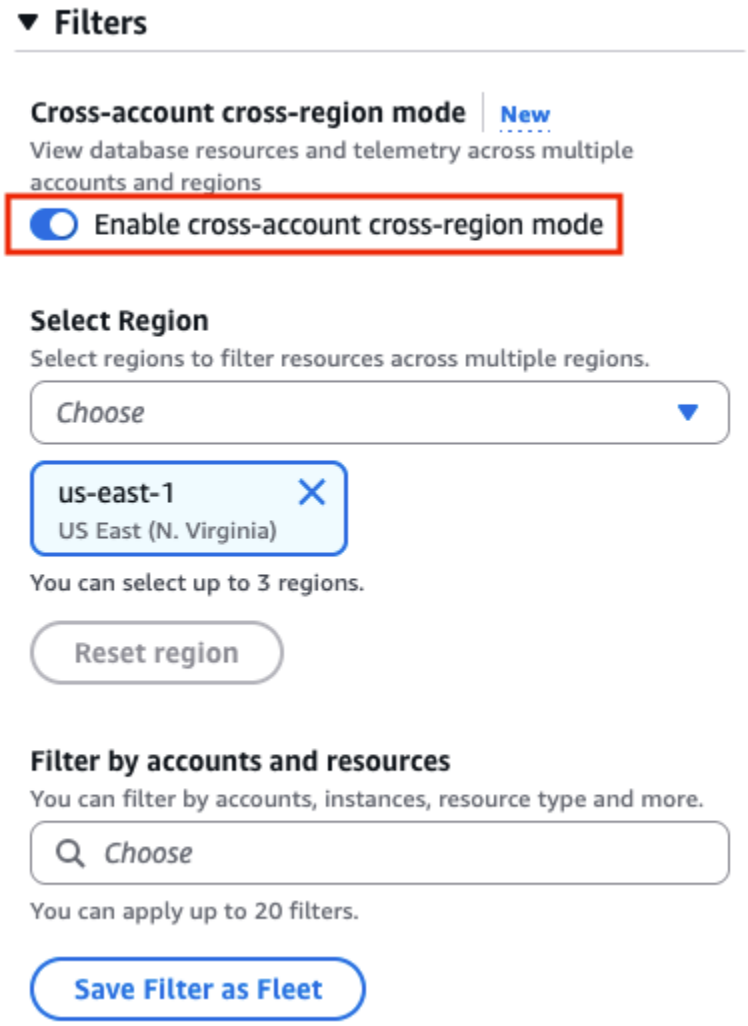
1. Follow the step-by-step guide in CloudWatch cross-account observability. CloudFormation templates are provided for infrastructure as code deployments.
2. In **Step 1: Set up a monitoring account** you need to at least select the following data types to share with your monitoring account for Database Insights to work across accounts and regions: **Logs**, **Metrics**, **Traces**, and **Application Signals - Services, Service Level Objectives (SLOs)**.
3. You need to do this process for **all regions** where you want to setup Database Insights with cross-account cross-region support.

Set-up Database Insights for cross-account cross-region console

1. Follow the step-by-step guide in Cross-account cross-Region CloudWatch console.
2. In **Step 5: Permissions**, you need to at least select the following data types to share with your monitoring account: **Include CloudWatch automatic dashboards**, and **Include read-only access for Database Insights.** Otherwise, you can also opt for **Full read-only access to everything in your account**, which will include all data sources available.
3. CloudWatch cross-region console is a global setting, so you only need to do this step once and it will work for all regions.

Using Database Insights cross-account cross-region dashboards

After you set up Database Insights for cross-account cross-region support, you can enable "Cross-account cross-region mode" in the **Filters** section of the left panel, right under **Database Views**. You will see a toggle button located on the top left corner, like in the screenshot below.



Once cross-account cross-region mode is enabled, new Filters are available, which allow you to select multiple regions and to filter by AWS Account IDs and labels, database resource types, and database resource identifiers.

By default, your current region and all accounts are selected. If you change your selection of regions and accounts, the Fleet Health Dashboard updates automatically to show resources that match the selected region and account filters.

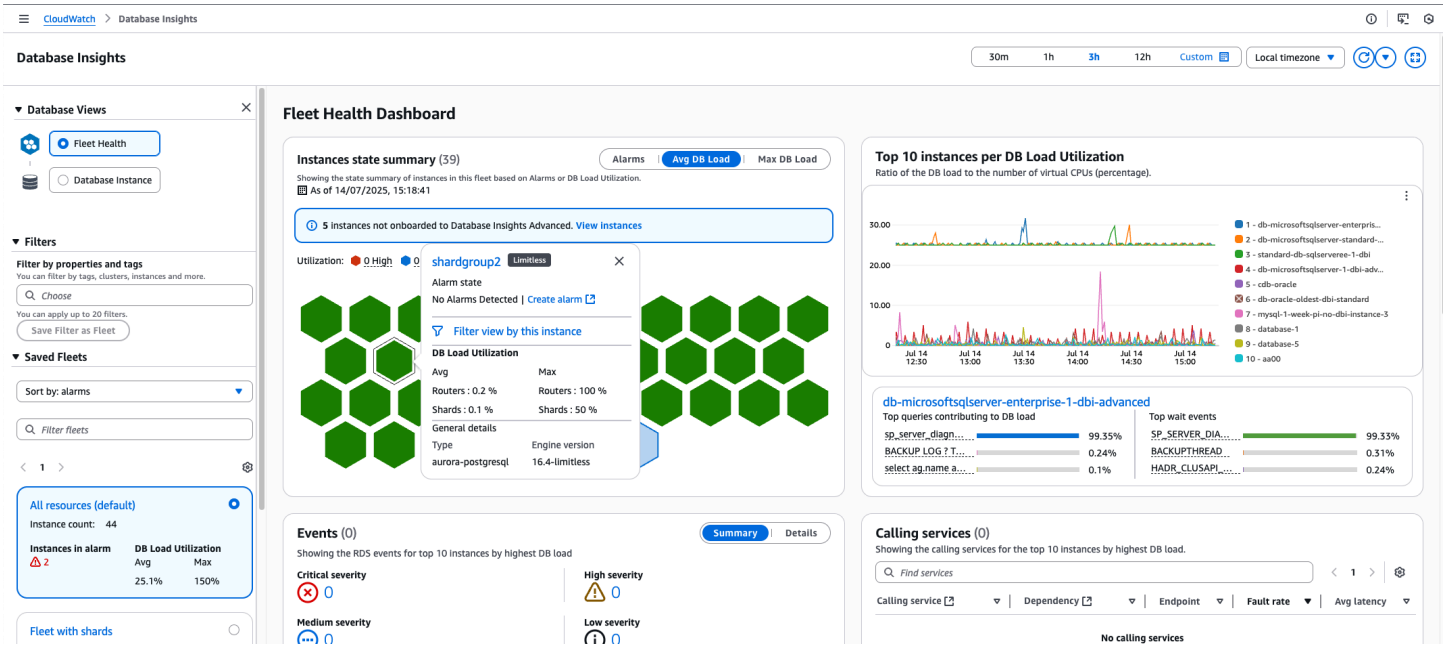
From the Fleet Health Dashboard, you can easily navigate to the Database Instance Dashboard for instances that belong to other AWS accounts and regions.

There is a few aspects you must take into consideration when working with Database Insights cross-account cross-region monitoring:

- Alarms must be created in the monitoring account only. Alarms can be configured on metrics from source accounts, but they need to be created in the monitoring account.
- Fleet monitoring views must be defined and saved in the monitoring account only.
- Custom metrics dashboards in Database Insights Instance Dashboard must be customised in the monitoring account only.
- The maximum number of regions allowed in the Database Insights Fleet Health Dashboard at a given time is 3.
- Only read operations are allowed from the Monitoring account. This means you cannot create performance analysis reports from the Monitoring account.
- When cross-account cross-region mode is enabled in the Fleet Health Dashboard, filtering by AWS Resource Tags and resource properties is not available.

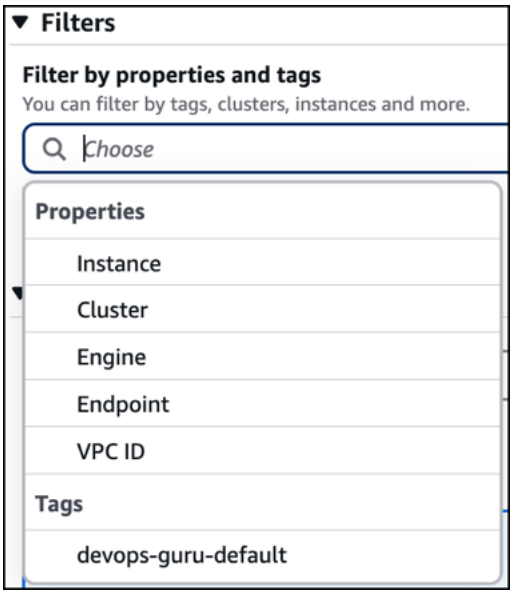
Viewing the Fleet Health Dashboard for CloudWatch Database Insights

You can use the Fleet Health Dashboard to view a snapshot of the health of your database fleet.



Fleet health views

A *database fleet* in Database Insights is a group of databases that you want to monitor. You can create a monitoring view for a database fleet by choosing filters in the **Filters** component. This component allows you to apply filters on properties, such as cluster or instance names and tags. In the Fleet Health Dashboard, CloudWatch shows databases that match at least one of the filter conditions for the fleet health view.

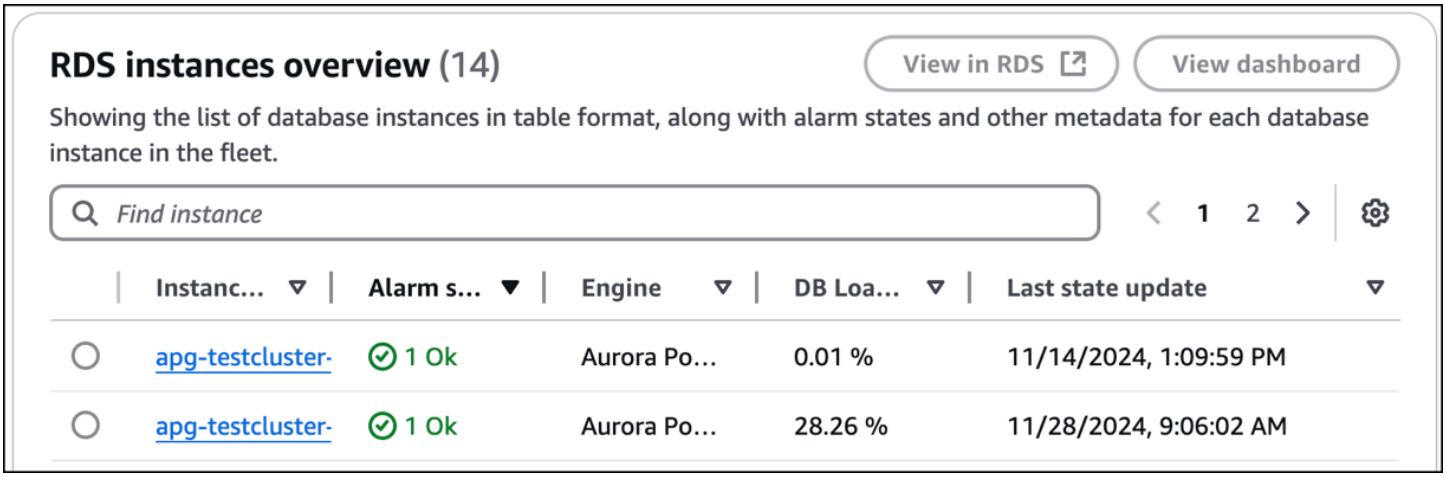


To create, modify, or delete views for database fleets, use the procedures in the following topics.

- Create a fleet health view for CloudWatch Database Insights
- Edit a fleet health view for CloudWatch Database Insights
- Delete a fleet health view for CloudWatch Database Insights

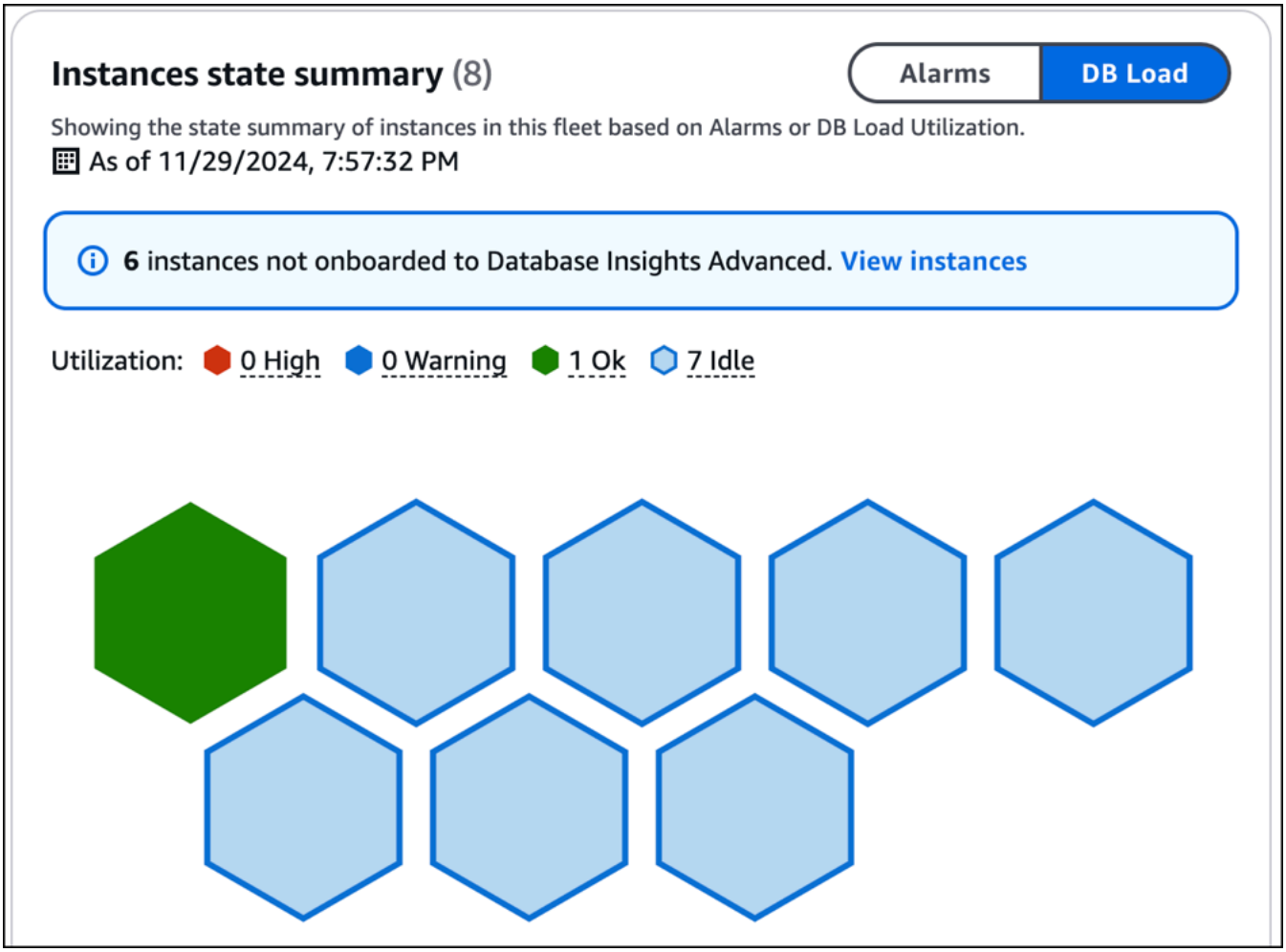
RDS instances overview table

Use the **RDS instances overview** table to view the alarm state, max DB Load percentage, and the time of the last state update for each instance in your fleet.

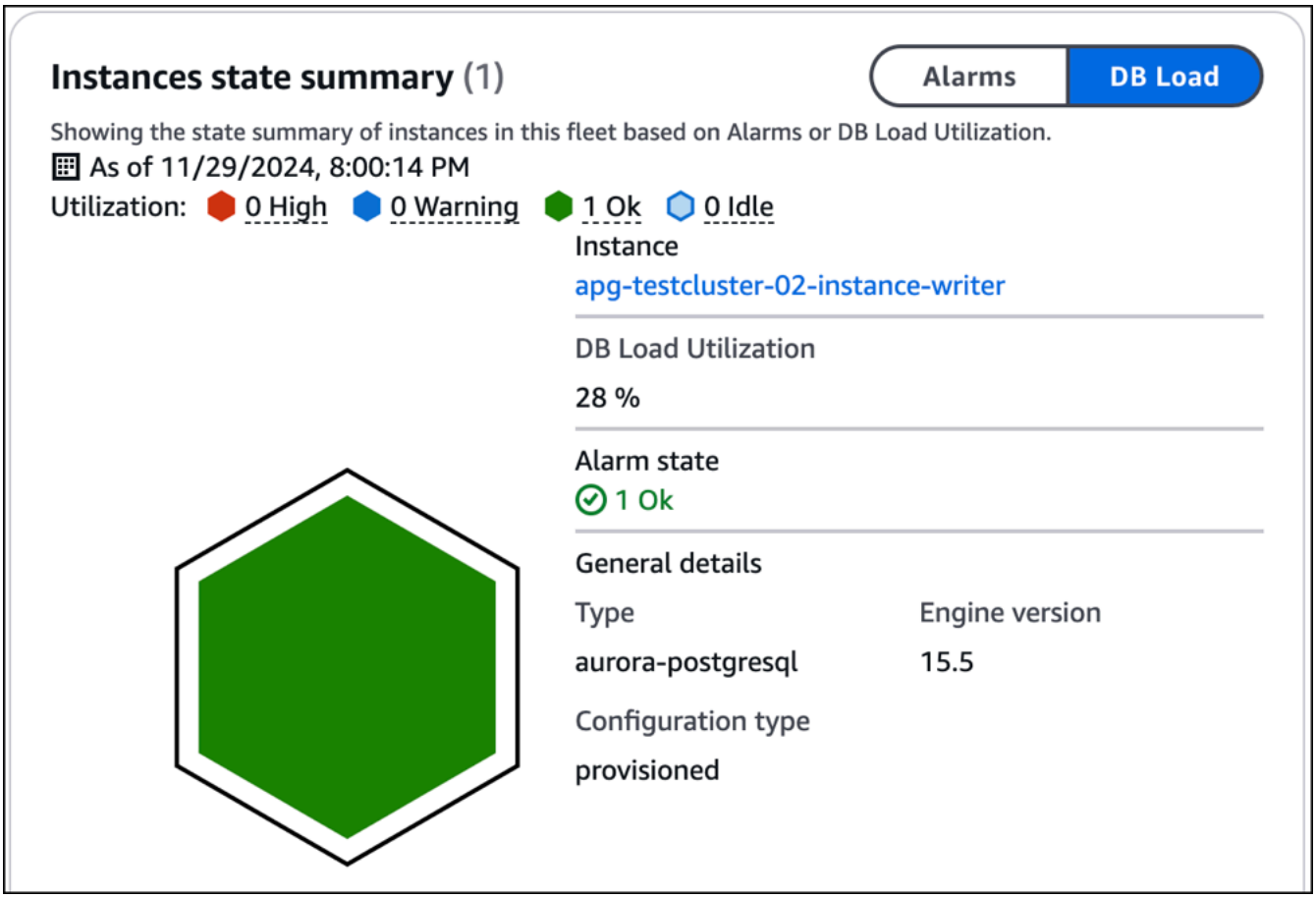


Instances state summary

Use the **Instances state summary** to view the health of all instances in your fleet. The Instances state summary provides two views based on **Alarms** and the DB Load metric. By default, CloudWatch displays the **Alarms** view.



Each node in the honeycomb represents an instance. For more information about an instance, you can choose the corresponding node and choose **Filter view by this instance**.



The honeycomb component summarizes the alarm state for instances in your fleet with the number of nodes in each state at the top of the honeycomb. CloudWatch displays the time of the last refresh of the data shown in the honeycomb.

When you switch to the **DB Load** view, you can see the overall health of the fleet from the point of view of the DB Load metric. Database load (DB Load) measures the number of active sessions in your database. DB Load is the key metric in Database Insights and is collected every second. CloudWatch categorizes DB instances into the following states based on thresholds for DB Load.

- High
- Warning
- Ok
- Idle

You can see the thresholds for DB Load by choosing the corresponding state icons.

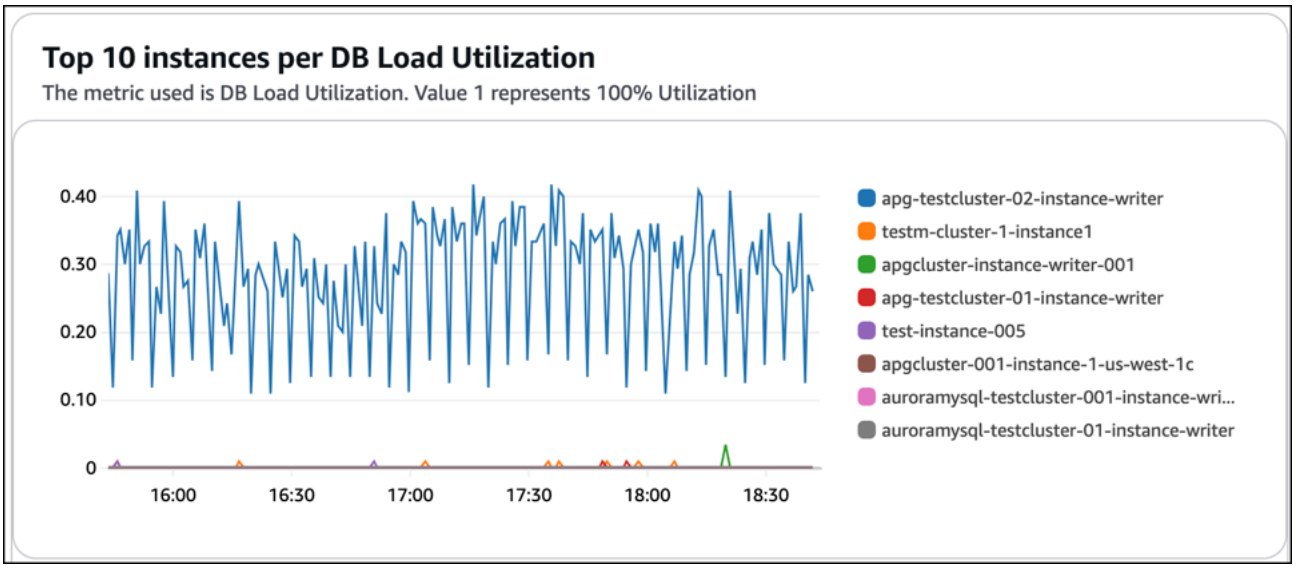
For information about DB Load for Amazon RDS, see Database load in the *Amazon RDS User Guide*. For information about DB Load for Amazon Aurora, see Database load in the *Amazon Aurora User Guide*.

By default, CloudWatch displays the average DB Load. Choose **Max** to monitor the maximum DB Load for each instance.

Choose a node from the Instances state summary to display alarms and DB Load for the instance.

Top 10 charts

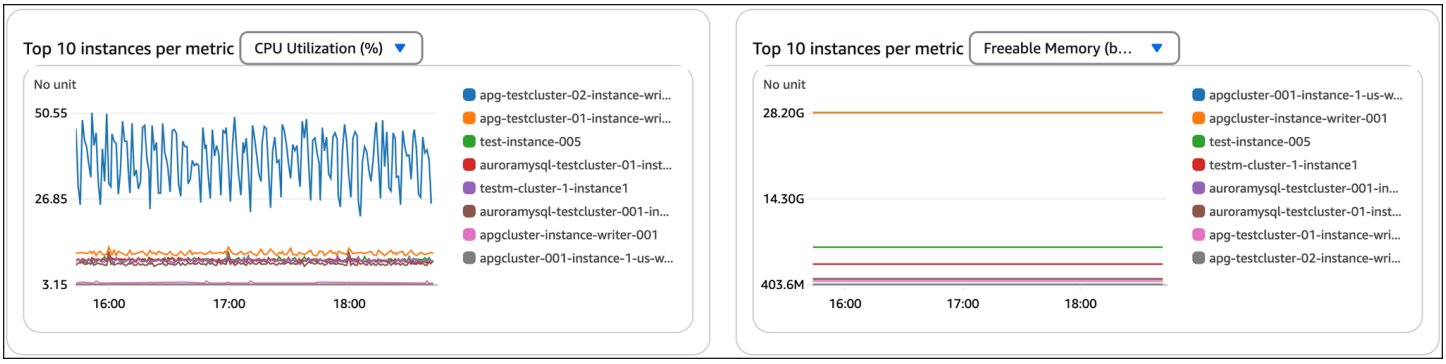
Use the **Top 10 instances per relative DB Load** chart to view the DB Load trend over time for the 10 instances with the highest DB Load. The chart also provides the top queries and top wait events for the instance with the highest DB Load.



Use the **Top 10 instances per metric** charts to compare two key metrics for the top 10 instances in your fleet. You can select the following metrics.

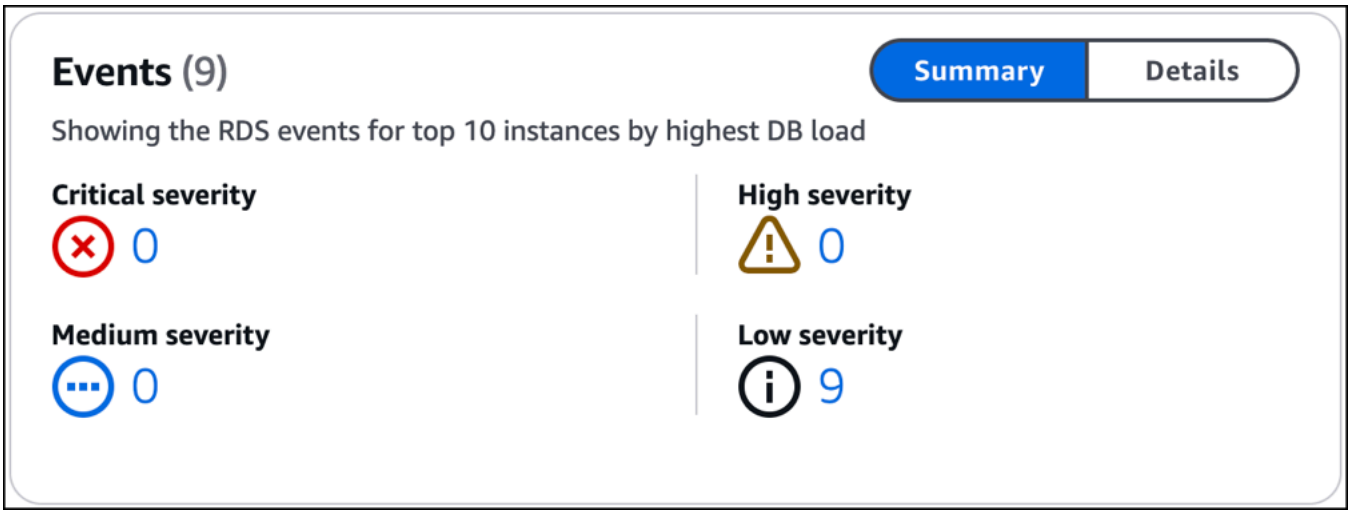
- CPU Utilization (%)
- Freeable Memory (%)
- DB Connections (%)
- Network throughput
- Read IOPS
- Write IOPS

- Read Latency
- Write Latency

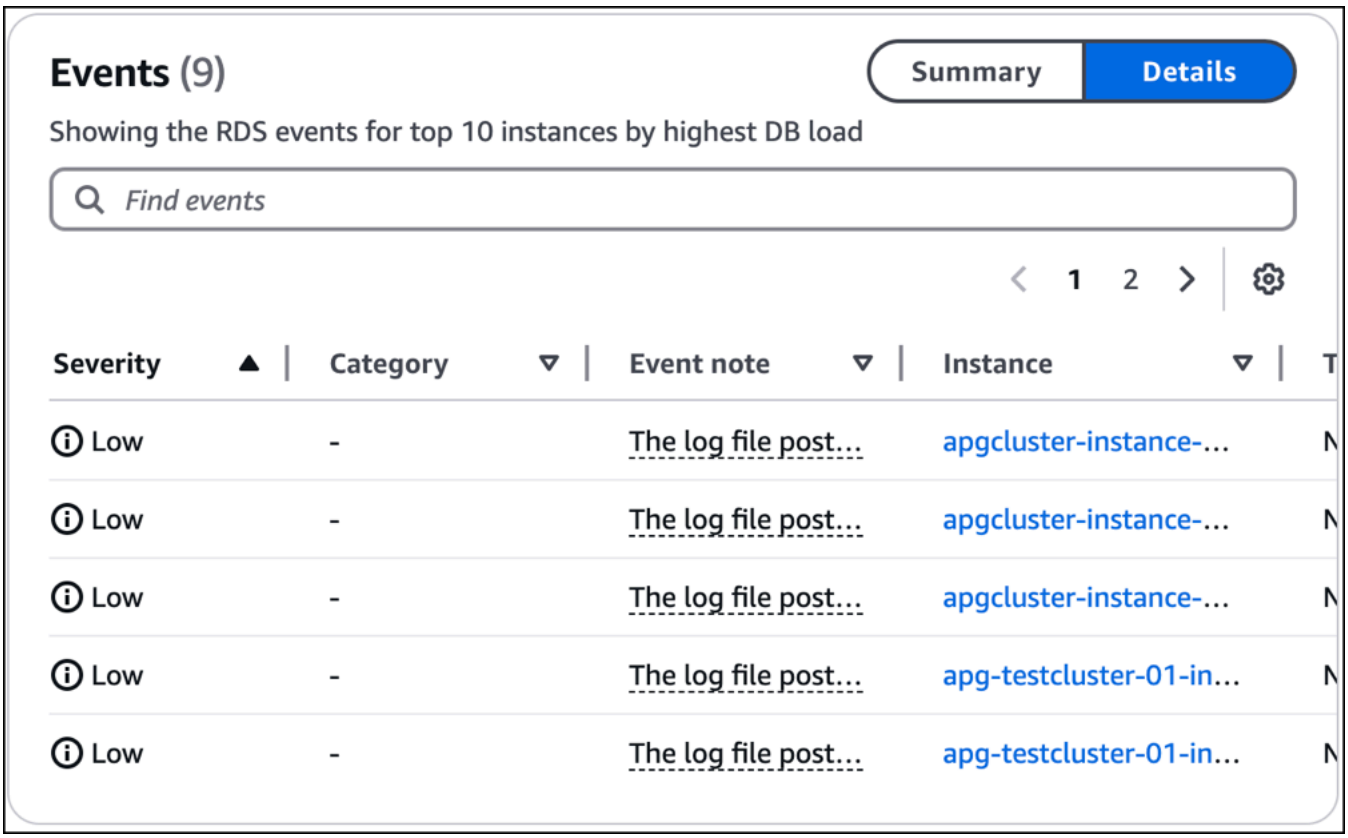


Amazon RDS events

Use the **Events** summary and table to view RDS events for instances in your fleet.



To view the **Events** table, choose **Details**.



For a list of events for Amazon RDS and Amazon Aurora, see the following topics.

- Amazon RDS event categories and event messages for Aurora in the *Amazon Aurora User Guide*
- Amazon RDS event categories and event messages in the *Amazon RDS User Guide*

Calling services table

Use the **Calling services** table to view CloudWatch Application Signals services that are calling your database endpoints and related application-level metrics such as latency or errors.



Database Insights shows the services that are calling your top 10 instances by DB Load. To view calling services for another instance, choose the instance in the database instance dashboard.

When the endpoint called by the application is an Aurora cluster, Database Insights will display either the writer or the reader endpoint for the Aurora cluster in the **Calling services** table, not the individual database instance. However, when the endpoint called by the application is an Amazon RDS cluster, Database Insights shows the specific database instance the application is calling within the Amazon RDS cluster.

For more information about CloudWatch Application Signals, see Application Signals.

Create a fleet health view for CloudWatch Database Insights

To create a fleet health view, use the following procedure.

To create a fleet health view

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose filters for the fleet of databases you want to monitor.
5. Choose **Save filter as fleet**.
6. In the **Save filter set (fleet)** window, enter a name for your fleet.
7. Choose the **Save** button.

To access the saved fleet, choose the **Saved fleets** dropdown. Then, choose a fleet.

Edit a fleet health view for CloudWatch Database Insights

To edit a fleet health view, use the following procedure.

To edit a fleet health view

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Saved fleets** dropdown.
5. Choose the vertical ellipsis for a fleet health view you want to edit.
6. In the **Edit filter set (fleet)** window, you can edit the name of the fleet and the filters for the fleet.
7. Choose the **Save** button.

Delete a fleet health view for CloudWatch Database Insights

To delete a fleet health view, use the following procedure.

To delete a fleet health view

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Saved fleets** dropdown.
5. Choose the vertical ellipsis for a fleet health view you want to delete.
6. In the **Delete filter set (fleet)** window, choose the **Delete** button.

Viewing the Database Instance Dashboard for CloudWatch Database Insights

Use the Database Instance Dashboard to view a snapshot of the health of a DB instance.

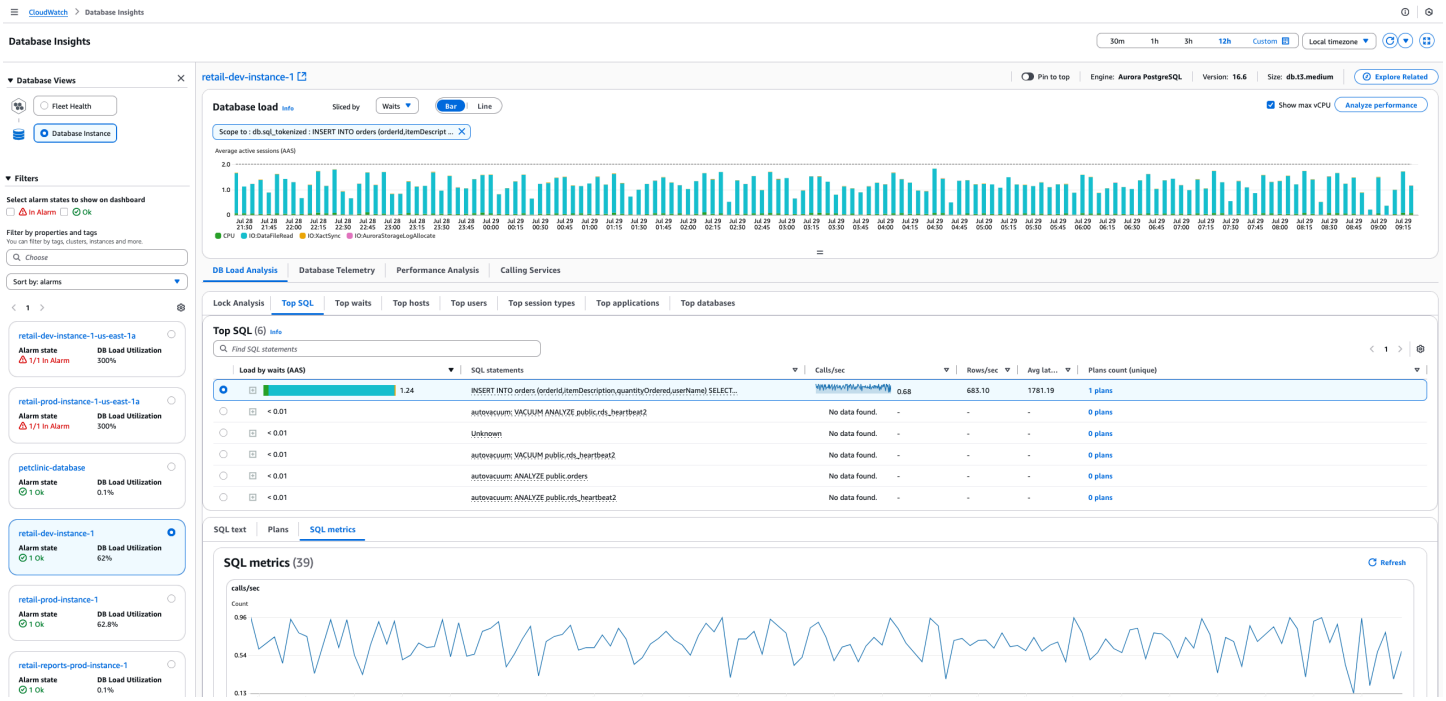
To analyze lock trees and execution plans for Amazon Aurora PostgreSQL, see the following topics.

Topics

- Analyzing lock trees for Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL with CloudWatch Database Insights
- Analyzing execution plans with CloudWatch Database Insights

Database load chart

Database load (DB Load) measures the level of session activity in your database. DB Load is the key metric in Database Insights, and Database Insights collects DB Load every second.



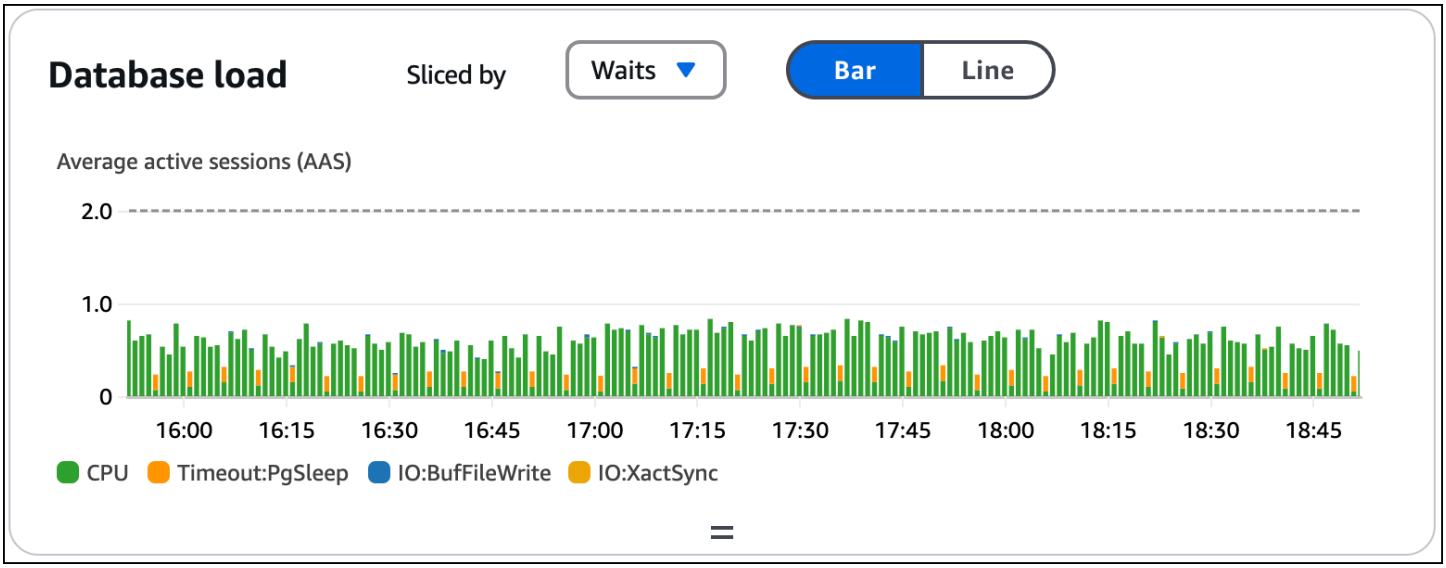
For more information about DB Load, see Database load in the *Amazon RDS User Guide* or Database load in the *Amazon Aurora User Guide*.

Use the **Database load** chart to view DB Load sliced (grouped) by the following dimensions for all supported database engines.

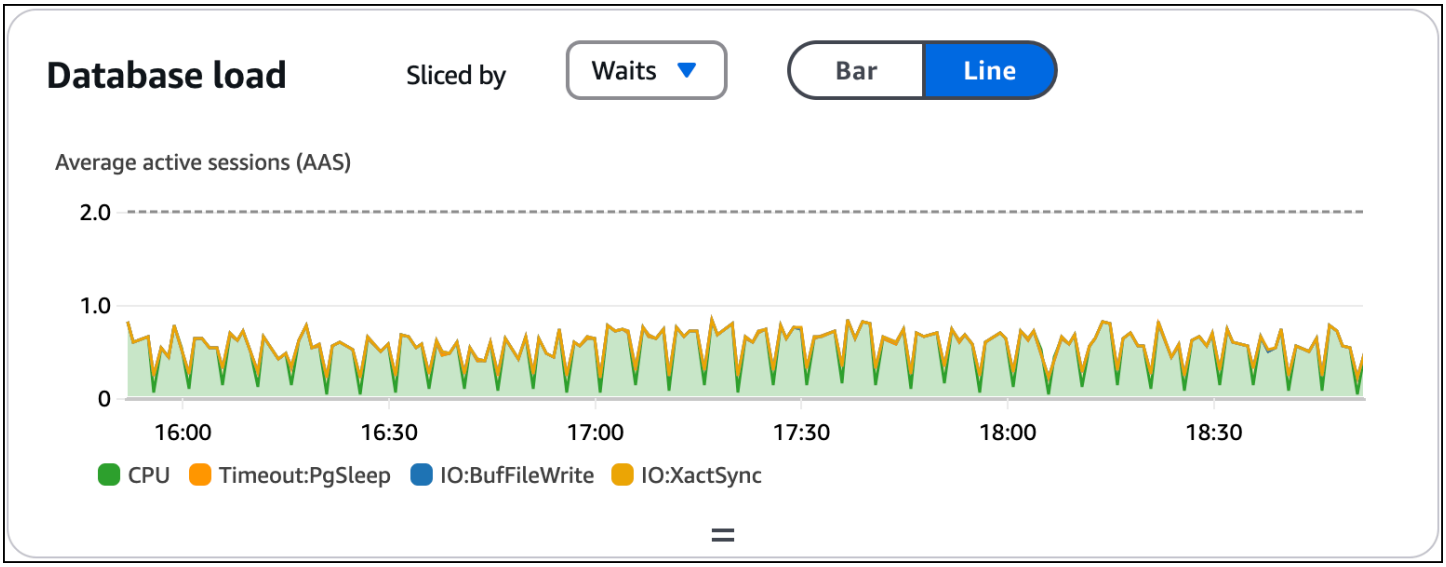
- Blocking object (only for database engines that support locking analysis)
- Blocking session (only for database engines that support locking analysis)
- Blocking SQL (only for database engines that support locking analysis)
- Database
- Host

- SQL
- User
- Waits
- Application (only for Amazon Aurora PostgreSQL)
- Plans (only for database engines that support execution plan capture)
- Session type (only for Amazon Aurora PostgreSQL)

> **Note**
>
> For information about analyzing Oracle PDB load in Amazon RDS, see Analyzing top Oracle PDB load in the *Amazon RDS User Guide*.



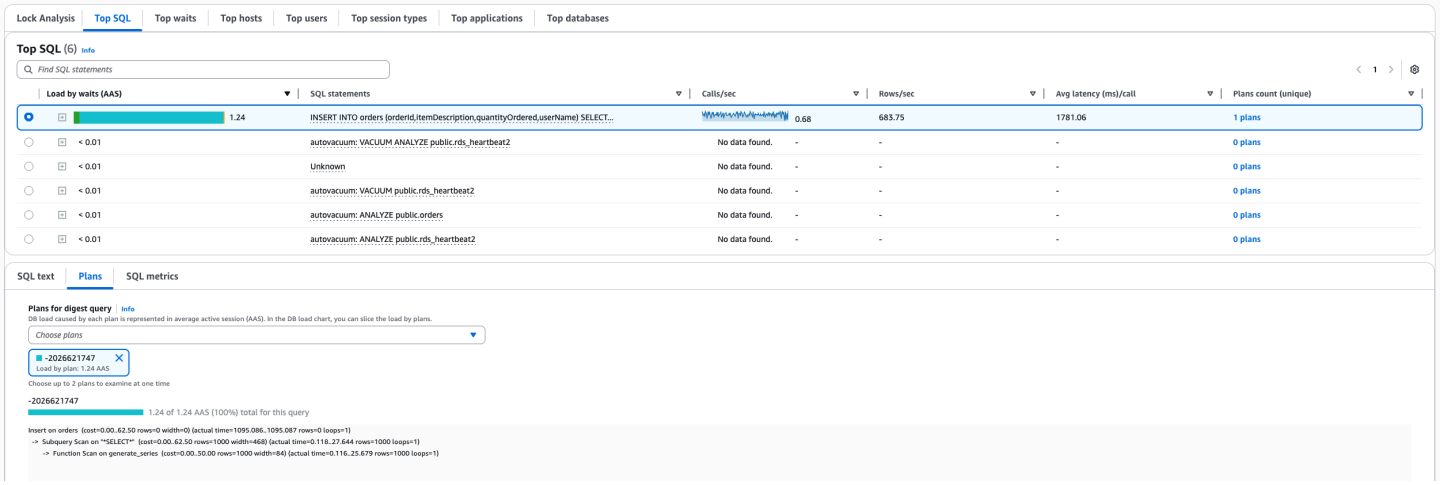
By default, CloudWatch displays DB Load with a bar chart. Choose **Line** to display DB Load with a stacked line chart.



DB Load analysis tab

Use the **DB Load analysis** tab to monitor the top contributors to DB Load for each of the following dimensions.

- Database
- Host
- SQL
- User
- Waits
- Lock analysis (only for database engines that support locking analysis)
- Application (only for Amazon Aurora PostgreSQL)
- Session type (only for Amazon Aurora PostgreSQL)

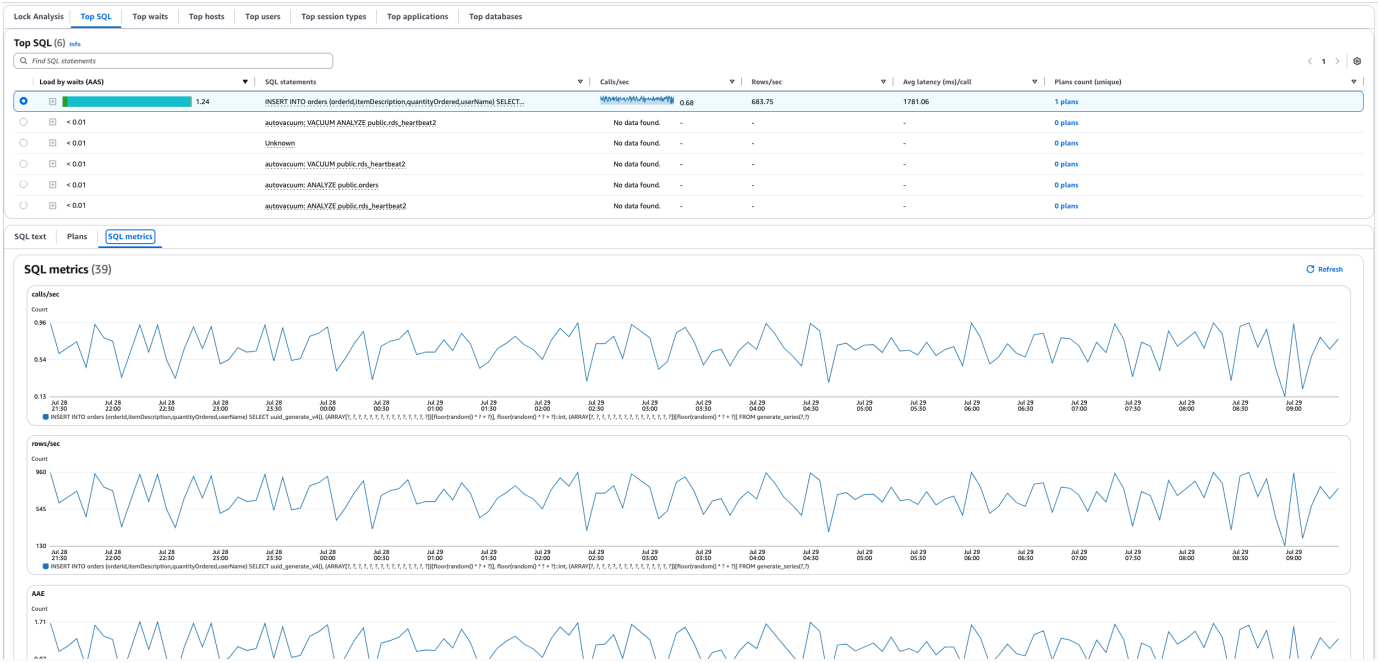


Analyze statistics for a query

You might want to analyze statistics for a query with a high DB Load. To analyze statistics for a query, use the following procedure.

To analyze statistics for queries

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **Top SQL** tab.
7. To view statistics for a query, choose a query.



Database telemetry tab

Use the **Database telemetry** tab to view metrics, logs, events, and slow queries for the selected instance.

Metrics section for database telemetry

The **Metrics** section displays a default metrics dashboard customized for each engine type.

You can customize this dashboard by adding OS metrics, database counter metrics, and CloudWatch metrics to it. You can also remove metrics from the dashboard. You can customize one dashboard for each engine type in a Region in your account. This means that all instances for a specific engine type in that Region in the same account will have the same metrics dashboard.

Users who have edit permissions for your dashboards in your account can edit any dashboard for any engine.

Changes you make to a dashboard are saved automatically, and apply to every instance of the database engine in that Region and account.



To customize the dashboard in the Database telemetry tab for an engine type

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**, **Database Insights**.
3. For **Database Views**, choose **Database Instance**.
4. In the **Filters** section, find and choose the database instance that you want to view metrics for.
5. Choose the **Database Telemetry** tab, then choose the **Metrics** tab.

   The default database instance dashboard appears.
6. To add a widget to the dashboard, do the following:

   a. Choose **Add Create widget**.

   b. In the **Create widget** popup, find the metric or metrics that you want to add, and select the checkbox for each one. If you select multiple metrics in this step, they will all appear in the same new widget on the dashboard. Then choose **Create widget**.

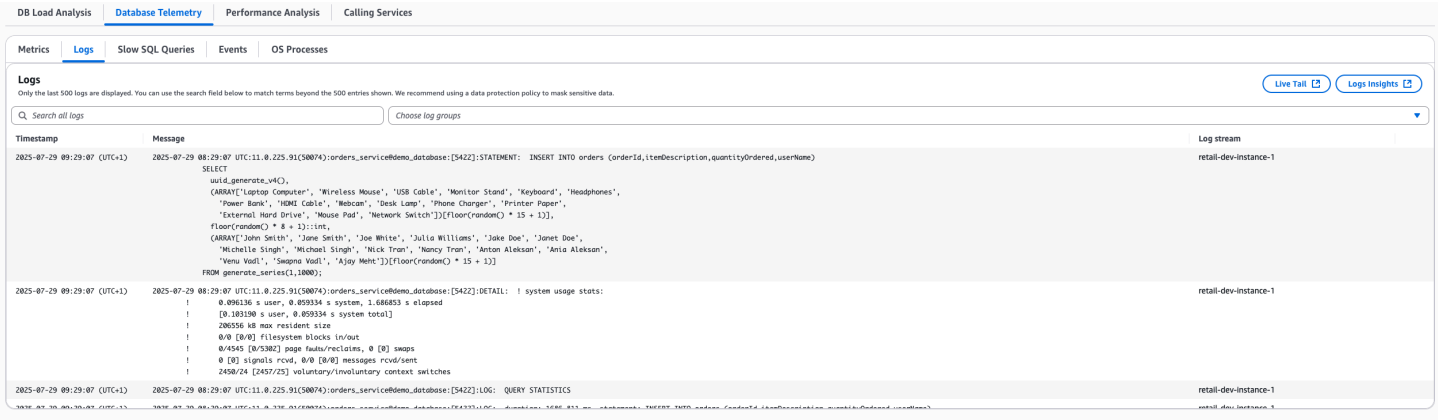
      Remember that any changes you make to this dashboard will apply to all Database Insights dashboards for this engine type in this Region in the account.
7. To delete a graph from the dashboard, choose the vertical ellipsis in the widget, then choose **Delete**.
8. To add more metrics to an existing widget in the dashboard, or change its title, choose the vertical ellipsis in the widget, and choose **Edit**. Then in the **Update widget** popup, find the

metric or metrics that you want to add, select their checkboxes, and choose **Update Widget**. You can also change the widget title.

9. After customizing a dashboard, you can reset it to its original default state by choosing **Reset Dashboard**.

Logs section for database telemetry

The **Logs** section provides a view of database logs exported to CloudWatch Logs for the selected DB instance.



For information about publishing logs to CloudWatch Logs for Amazon RDS, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon RDS User Guide*. For information about publishing logs to CloudWatch Logs for Amazon Aurora, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon Aurora User Guide*.

For Aurora PostgreSQL Limitless Databases, logs are automatically published to CloudWatch Logs and are discoverable in the Database Insights console.

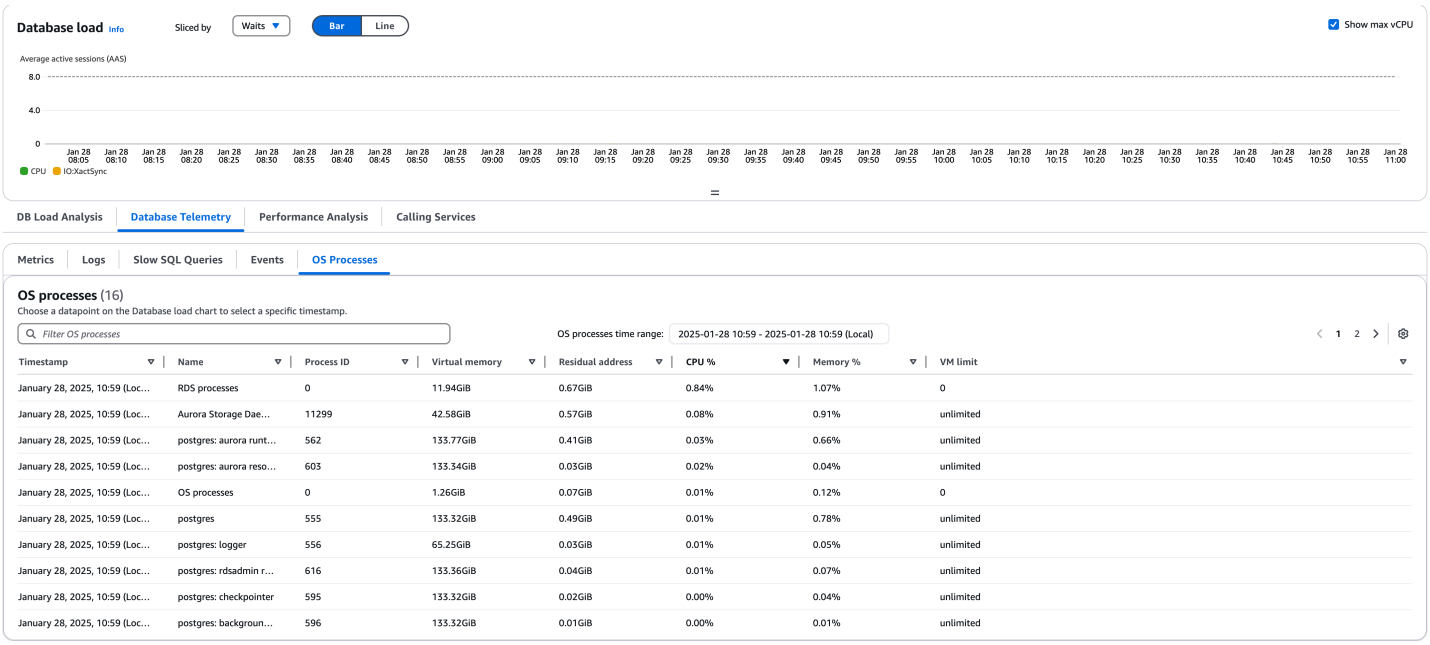
OS processes data for database telemetry

You can use the **OS Processes** tab within the **Database telemetry** tab to view metrics for the operating system (OS) that your DB instance runs on. The metrics provide a snapshot of OS processes running on your databases for a given timestamp, as well as key metrics such as memory and CPU utilization for each running process. Database Insights correlates these metrics with the metrics in your database load chart, so if you choose a data point in the database load chart, the OS processes data is updated to display telemetry from the same time stamp.

When you choose a data point, Database Insights automatically select the period to display, depending on the time range you have chosen for the overall page. The farthest back that you can go depends on the retention time that you have configured for the `RDSOSMetrics` log group.

If you haven't chosen a time stamp, by default the table is populated with telemetry for the latest timestamp.

> **Note**
>
> OS process information is available only if you have Amazon RDS Enhanced Monitoring enabled. Enhanced Monitoring incurs additional charges. For more information, see Cost of Enhanced Monitoring. Enhanced Monitoring is automatically enabled for Aurora PostgreSQL Limitless Databases.



In the **OS processes** view, the following data is displayed for each process:

- **Process ID** –The ID of this process.
- **Virtual memory** –The amount of virtual memory allocated to the process, in Kibibytes.
- **Residual address** – The actual physical memory being used by the process.
- **CPU %** – The percentage of the total CPU bandwidth being used by the process.
- **Memory %** – The percentage of the total memory being used by the process.

- **VM limit** – The maximum amount of virtual memory that can be allocated to the process.

  If the value in this column is 0, then VM limits are not applicable to that process.

The monitoring data that is displayed is retrieved from Amazon CloudWatch Logs. You can also retrieve these metrics directly from the log stream in CloudWatch Logs. For more information, see Viewing OS metrics using CloudWatch Logs.

OS processes metrics are not returned during the following:

- A failover of the database instance.
- Changing the instance class of the database instance (scale compute).

OS processes metrics are returned during a reboot of a database instance because only the database engine is rebooted. Metrics for the operating system are still reported.

Slow SQL Queries section for database telemetry

To view slow SQL queries and query patterns, you must enable log exports to CloudWatch Logs and configure DB parameters for your database.

For information about publishing Amazon RDS logs to CloudWatch Logs, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon RDS User Guide*.

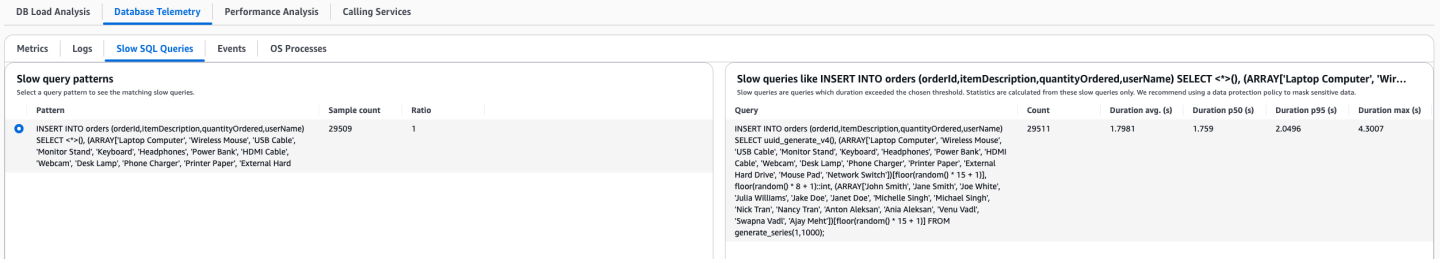
For information about publishing Aurora logs to CloudWatch Logs, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon Aurora User Guide*.

For information about configuring DB parameters for your database in Amazon RDS, see Configuring your database to monitor slow SQL queries with Database Insights for Amazon RDS in the *Amazon RDS User Guide*.

For information about configuring DB parameters for your database in Amazon Aurora, see Configuring your database to monitor slow SQL queries with Database Insights for Amazon Aurora in the *Amazon Aurora User Guide*.

The **Slow SQL Queries** section provides a list of slow query patterns sorted by frequency. By selecting a pattern, you can view a list of slow queries that match the selected pattern. You can use the slow query list to identify slow queries affecting your DB instance.

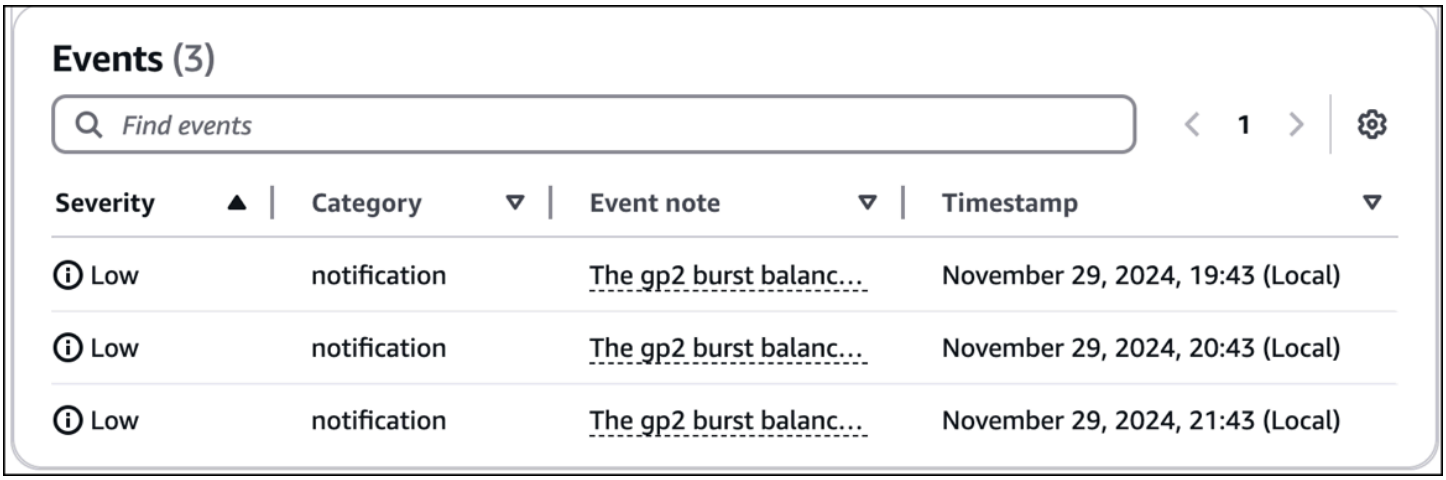
Database Insights displays statistics for slow queries. The statistics represent only queries that exceed the configured slow query duration threshold.



> **Important**
>
> Slow queries may contain sensitive data. Mask your sensitive data with CloudWatch Logs. For more information about masking log data, see Help protect sensitive log data with masking in the *Amazon CloudWatch Logs User Guide*.

Events table

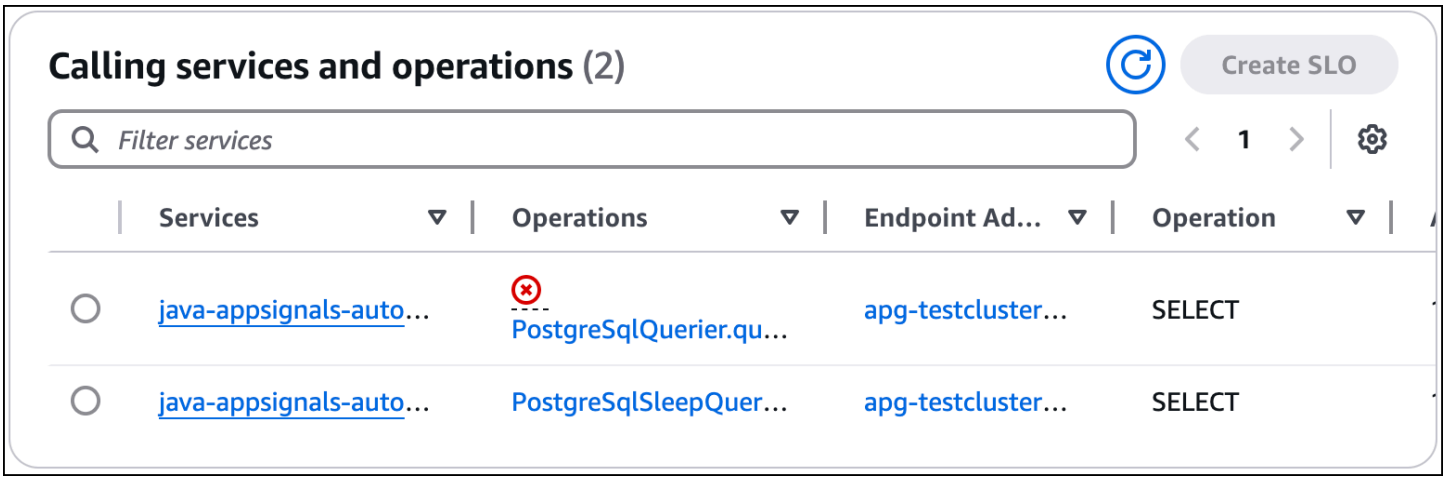
Use the **Events** table to view RDS events for your DB instance. For a list of events for Amazon Aurora, see Amazon RDS event categories and event messages for Aurora in the *Amazon Aurora User Guide*. For a list of events for Amazon Relational Database Service, see Amazon RDS event categories and event messages for Aurora in the *Amazon RDS User Guide*.



Calling services tab

Database Insights shows the services and operations that are calling your instance. Database Insights integrates with CloudWatch Application Signals to provide metrics for each service and operation, including availability, latency, errors, and volume.

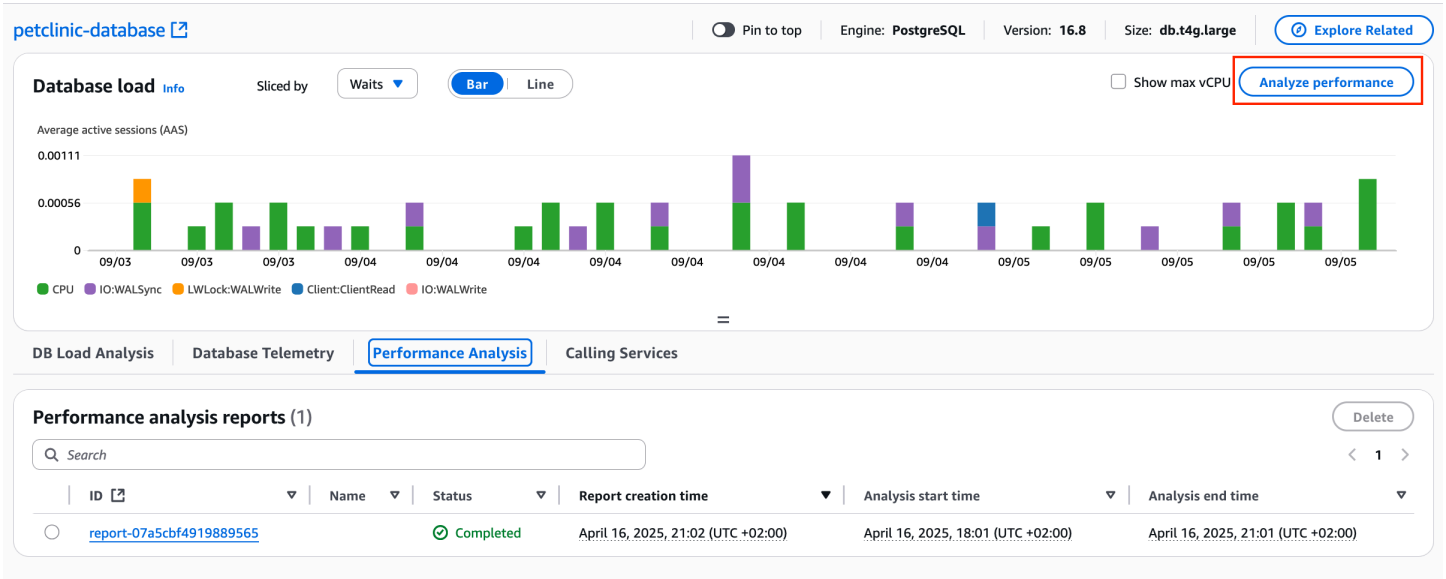
When the endpoint called by the application is an Aurora cluster, Database Insights will show either the writer or the reader endpoint for the Aurora cluster in the **Calling services** table, not the individual database instance. However, when the endpoint called by the application is an Amazon RDS cluster, Database Insights shows the specific database instance the application is calling within the Amazon RDS cluster."



Analyze database performance on demand with CloudWatch Database Insights

Analyze database performance with on-demand analyses for your Amazon RDS databases with CloudWatch Database Insights.

You can run on-demand performance analysis by selecting **Analyze Performance** on the top right corner of the **Database load** chart. The report will run for the selected time period. Use the **Performance analysis** tab to view performance analysis reports for databases in your fleet.



For information about performance analysis reports for Amazon Aurora, see Analyzing database performance for a period of time in the *Amazon Aurora User Guide*.

Integrating CloudWatch Database Insights with CloudWatch Application Signals

Integrate CloudWatch Database Insights with CloudWatch Application Signals.

Use the **Calling services** tab to view the CloudWatch Application Signals services and operations that called an endpoint of the selected instance. By default, CloudWatch sorts the table by fault rate. Choose values in the **Services**, **Operations**, or **Endpoint address** columns to view the corresponding resource in the CloudWatch Application Signals console.

For more information about supported systems for CloudWatch Application Signals, see Supported systems.

Analyzing lock trees for Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL with CloudWatch Database Insights

To troubleshoot performance issues caused by locks, you can analyze lock trees for Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL databases with CloudWatch Database Insights using the following.

- **Sliced by** dropdown – Choose the **Blocking object**, **Blocking session**, or **Blocking SQL** dimensions in the **Database load** chart to view how distinct top blockers contribute to DB Load over time. With the DB load chart, you can analyze if top blockers are constant or change often. Then, you can troubleshoot the blockers.



- **Lock analysis** tab – Choose **DB Load Analysis**, then choose the **Lock analysis** tab to view information about lock contention in your database.



> **Note**
>
> CloudWatch Database Insights supports lock analysis for all Aurora PostgreSQL versions. To analyze lock trees, you must have Database Insights Advance Mode enabled. For information on how to turn on Advanced mode, see Turning on the Advanced mode of Database Insights for Amazon Aurora and Turning on the Advanced mode of Database Insights for Amazon Relational Database Service

The lock analysis tab provides information about lock contention for your database. The lock tree visualization shows the relationships and dependencies between lock requests from different sessions.

Database Insights captures snapshots every 15 seconds. Snapshots show the lock data for your database at a point in time.

> **Note**
>
> When CloudWatch detects high locking, CloudWatch displays the **High locking detected** banner for the **Lock analysis** tab. CloudWatch detects high locking if CloudWatch takes a lock snapshot for each 15 second interval for 15 consecutive minutes.

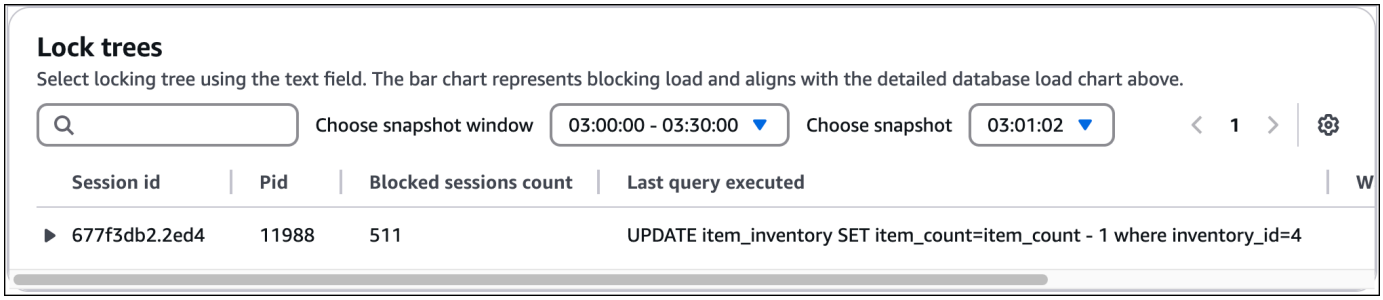
Each node in the tree represents a specific session. The parent node is a session that is blocking its child nodes.

To analyze lock trees, use the following procedure.

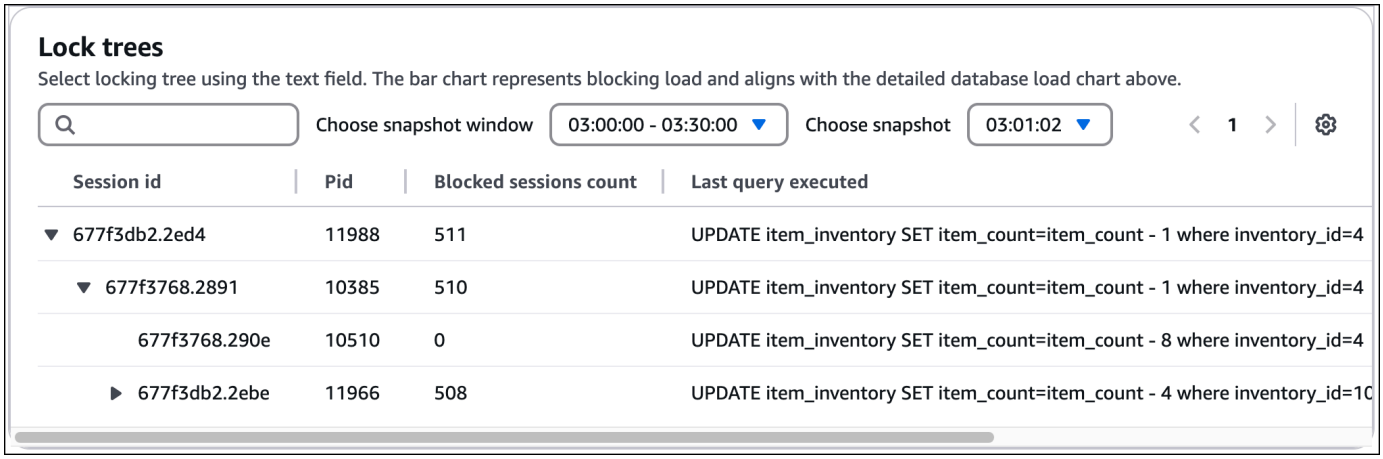
To analyze lock trees

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **DB load analysis** tab.
7. Choose the **Lock analysis** tab.

   To view lock data for a DB instance, choose a period of 1 day or less.
8. Choose a snapshot window. By default, Database Insights chooses the snapshot window with the most blocked sessions.



9. To view lock data for a snapshot, choose the time Database Insights took the snapshot.
10. To expand a lock tree, choose the arrow next to the session ID.



Lock snapshot data

Database Insights provides the following information for each lock request. To view columns that aren't enabled by default, choose the **Settings** icon for the **Lock trees** table and enable other columns.

Column name	Definition	Default column	Notes
`session_id`	The unique session identifier.	Yes	The `session_id` is derived from `HEX(pg_stat_activity.backend_start).HEX(pg_locks.pid)`.
`pid`	The PID of this backend.	Yes	`pg_locks.pid`
`blocked_sessions_count`	The number of sessions blocked by this lock.	Yes	The `blocked_sessions_count` is derived from the number of session IDs blocked by this lock.

Column name	Definition	Default column	Notes
last_query_executed	The last query executed by this session. For blockers, it may not be the query that holds the blocking lock.	Yes	pg_stat_activity.query
wait_event	The wait event name if the backend is currently waiting, otherwise the value is NULL.	Yes	pg_stat_activity.wait_event
blocking_time_(In Seconds)	The time (in seconds) since the start of this lock.	Yes	The blocking_time_(In Seconds) is derived from the start time of the waiting transaction (pg_locks.waitstart) for the first waiter.
blocking_mode	The lock mode held by the blocking session.	No	pg_locks.mode
waiting_mode	The lock mode requested by the waiting session.	No	pg_locks.mode
application	The name of the application that is connected to this backend.	No	pg_stat_activity.application_name
blocking_txn_start_time	The start time of the blocking transaction or null if no transaction is active.	No	pg_stat_activity.xact_start
waiting_start_time	The time when a waiting user session started waiting for this lock, or null if the lock is held.	No	pg_locks.waitstart

Column name	Definition	Default column	Notes
session_start_time	The time when a user session was started.	No	pg_stat_activity.backend_start
state	The state of a backend.	No	pg_stat_activity.state
wait_event_type	The type of wait event for which this session is waiting.	No	pg_stat_activity.wait_event_type
last_query_exec_time	The time when the last query was started.	No	pg_stat_activity.query_start
user	The name of the user logged into this backend.	No	pg_stat_activity.usename
host	The host name of the connected client, as reported by a reverse DNS lookup of client_addr . This field will only be non-null for IP connections, and only when log_hostname is enabled.	No	pg_stat_activity.client_hostname
port	The TCP port number that the client is using for communication with this backend, or -1 if a Unix socket is used. If this field is null, it indicates that this is an internal server process.	No	pg_stat_activity.client_port

Column name	Definition	Default column	Notes
client_address	The IP address of the client connected to this backend. If this field is null, it indicates either that the client is connected through a Unix socket on the server machine or that this is an internal process such as autovacuum.	No	pg_stat_activity.client_addr
granted	The value is true if lock is held and false if lock is awaited.	No	pg_locks.granted
waiting_tuple	The tuple number targeted by the lock within the page, or null if the target is not a tuple.	No	pg_locks.tuple
waiting_page	The page number targeted by the lock within the relation, or null if the target is not a relation page or tuple.	No	pg_locks.page
waiting_transaction_id	The ID of the transaction targeted by the lock, or null if the target is not a transaction ID.	No	pg_locks.transactionid
waiting_relation	The OID of the relation targeted by the lock, or null if the target is not a relation or part of a relation.	No	pg_locks.relation
waiting_object_id	The OID of the lock target within its system catalog, or null if the target is not a general database object.	No	pg_locks.objid
waiting_database_id	The OID of the database in which the lock target exists, or zero if the target is a shared object, or null if the target is a transaction ID.	No	pg_locks.database

Column name	Definition	Default column	Notes
waiting_database_name	The name of the database in which the lock target exists.	No	pg_stat_activity.datname
waiting_locktype	The type of the lockable object: relation, extend, frozenid, page, tuple, transactionid, virtualxid, spectoken, object, userlock, advisory, or applytransaction.	No	pg_locks.locktype
is_fastpath	The value is true if the lock was taken with the fast path and false if taken from the main lock table.	No	pg_locks.fastpath

For more information about the values in the pg_stat_activity and pg_locks views, see the following topics in the PostgreSQL documentation.

- pg_locks
- pg_stat_activity

Analyzing execution plans with CloudWatch Database Insights

You can analyze execution plans for the Amazon Aurora PostgreSQL, RDS for Microsoft SQL Server and RDS for Oracle databases by using the following methods.

- **Sliced by** dropdown – Choose the **Plans** dimension in the **Database load** chart to view how different plans contribute to DB Load over time.
- **Top SQL** tab – Choose **DB Load Analysis**, then choose the **Top SQL** tab to view the number of plans for each digest query.

  To analyze execution plans for a digest query, choose the query and then choose the **Plans** tab. For more information, see the following procedure.

Prerequisites

To analyze execution plans, you must be using the Advanced mode of Database Insights. For information on how to turn on Advanced mode, see Turning on the Advanced mode of Database Insights for Amazon Aurora and Turning on the Advanced mode of Database Insights for Amazon Relational Database Service.

If you are using Aurora PostgreSQL, you also have the following prerequisites:

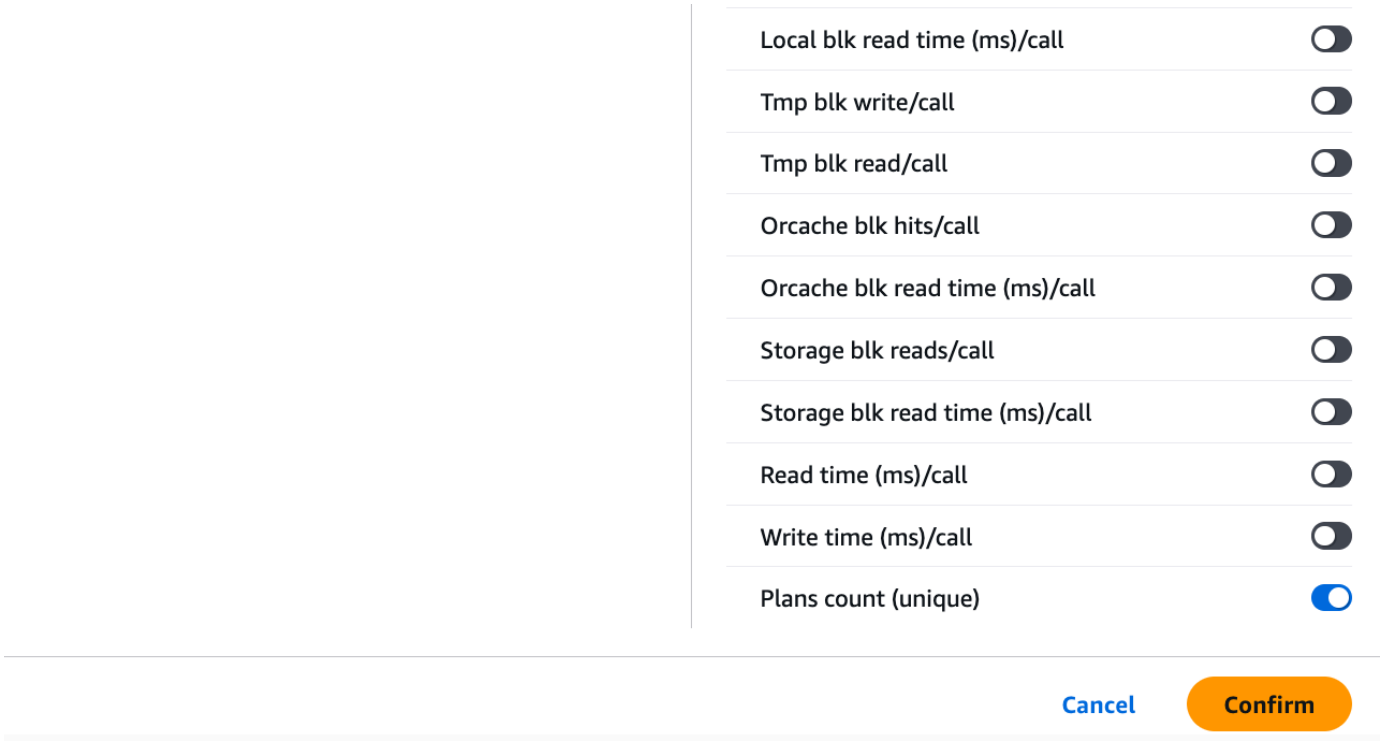
- Your DB instance must use Aurora PostgreSQL version 14.10, 15.5, or later. For information about upgrading your Aurora PostgreSQL DB cluster, see Upgrading Amazon Aurora PostgreSQL DB clusters in the *Amazon Aurora User Guide*.
- You must configure your DB cluster to analyze execution plans by setting the parameter `aurora_compute_plan_id` to on with one of the following options.
  - Creating a DB cluster parameter group in Amazon Aurora in the *Amazon Aurora User Guide*
  - Modifying parameters in a DB cluster parameter group in Amazon Aurora in the *Amazon Aurora User Guide*

Analyze execution plans

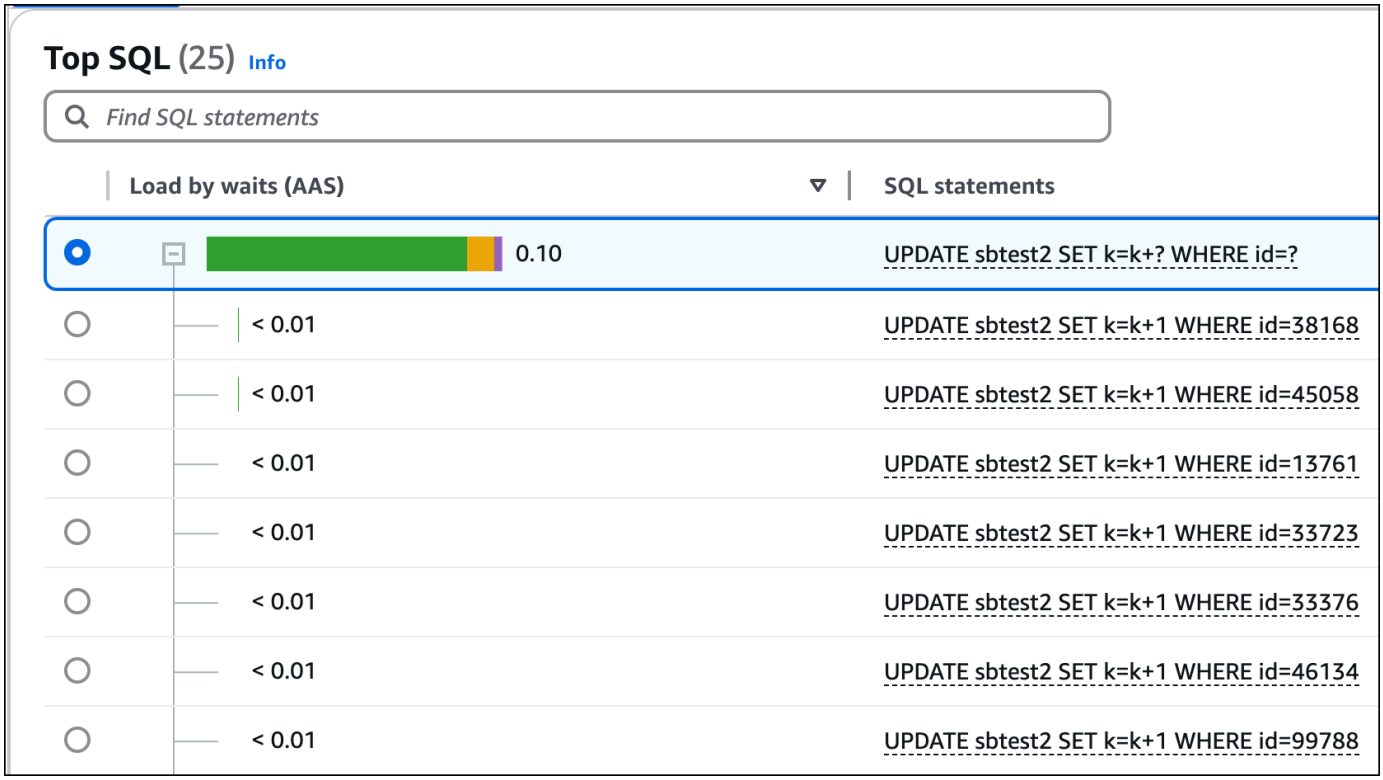
To analyze execution plans, use the following procedure.

To analyze execution plans

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Insights**.
3. Choose **Database Insights**.
4. Choose the **Database Instance** view.
5. Choose a DB instance.
6. Choose the **Top SQL** tab. The **Plans Count** column shows the number of plans collected for each digest query.
7. (Optional) If the **Plans Count** column doesn't appear, choose the **Settings** icon on the **Top SQL** table to customize the visibility and order of columns.



8. Choose a digest query to expand it into its component statements.

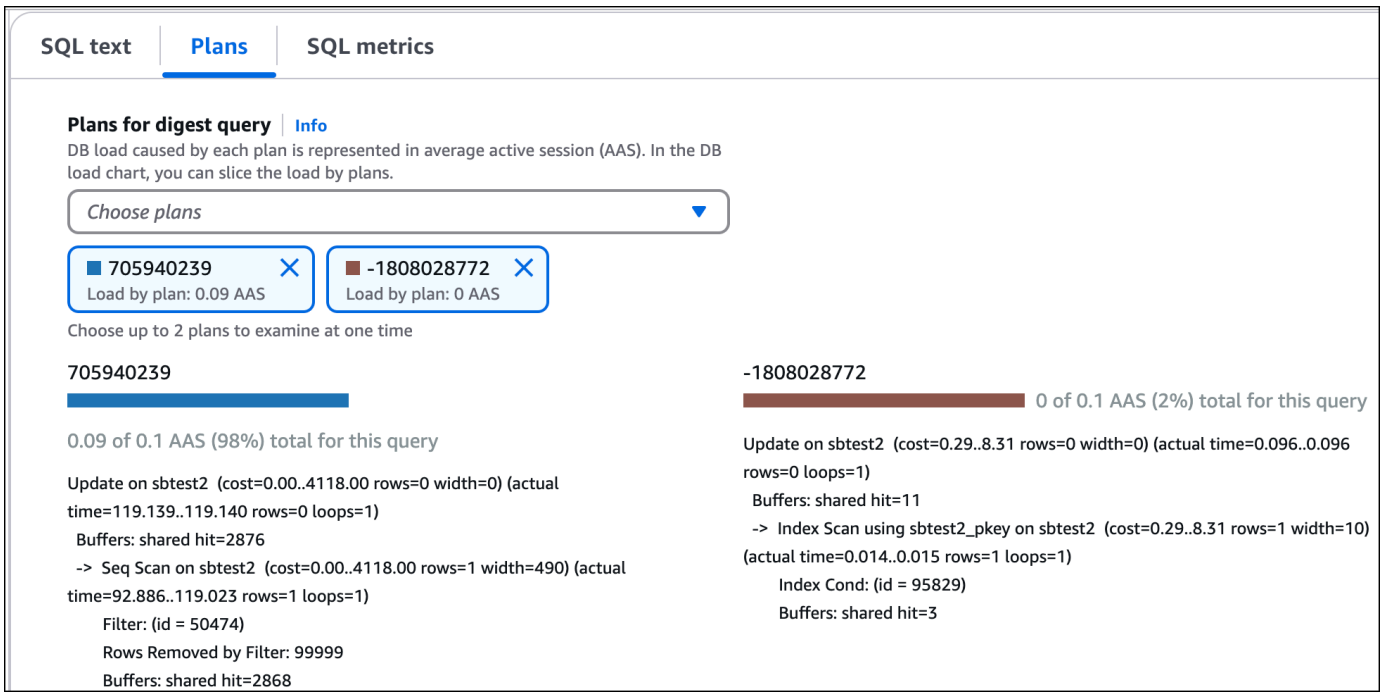


9. Scroll down and view the SQL text. Then, choose the **Plans** tab.

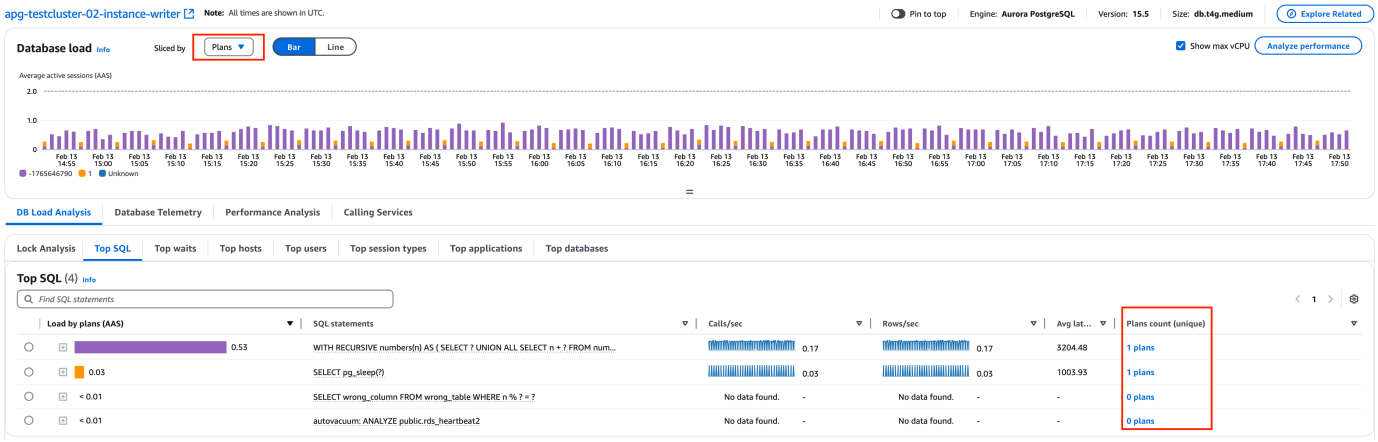
   By default, CloudWatch displays the estimated execution plan. For Aurora PostgreSQL, to view actual execution plans, enable the `aurora_stat_plans.with_analyze` parameter for your DB instance. For more information about the parameter `aurora_stat_plans.with_analyze`, see Monitoring query execution plans and peak memory for Aurora PostgreSQL in the *Amazon Aurora User Guide*.

10. To compare plans from the same digest query, choose two **Plans** from the **Plans for digest query** list.

    You can view either one or two plans for a query at a time. In the following example screenshot, both plans are for Aurora PostgreSQL.



11. You can also view how each plan contributes to DBLoad over time by choosing **Plans** in the **Slice by** drop-down in the DBLoad chart.



Monitoring Aurora Limitless databases with Database Insights

Database Insights supports monitoring Aurora PostgreSQL Limitless Databases at both the fleet and instance levels. Your Aurora PostgreSQL Limitless Databases are discoverable in both the Database Instance Dashboard and the Fleet Health Dashboard.

Aurora PostgreSQL Limitless Databases use *shard groups*. Each shard group consists of multiple database instances that work together to process distributed workloads. Database Insights helps you understand the load distribution among the instances within a shard group.

In the Fleet Health Dashboard, Database Insights provides monitoring of your Limitless shard groups together with the rest of databases that make up your database fleets. You can get an opinionated view of health and DBLoad utlilization for your Limitless shard groups in the same way you do for other databases in the fleet. In the Instance Dashboard, Database Insights provides monitoring at both the shard group level and for individual instances within the group. Database Insights provides a new view per shard group where you can see database load distributed across the instances in the shard group. From there, you can navigate to the specific instance dashboard within a shard group.

Available features for Aurora Limitless

The following table displays the features available for Aurora PostgreSQL Limitless databases. They indicate whether each feature is supported in the Standard and Advanced monitoring modes, and whether they are available at the Shard Group level, the instance level, and whether they are available in the Fleet or the Instance Dashboard of Database Insights.

Feature	Standard	Advanced	ShardGroup	Instance	Database Insights dashboard
Analyze the top contributors to DB Load by dimension	Supported	Supported	Yes	Yes	Instance
Query, graph, and set alarms on database metrics with up to 7 days of retention	Supported	Supported	Yes	Yes	Instance

Feature	Standard	Advanced	ShardGroup	Instance	Database Insights dashboard
Define fine-grained access control policies to restrict access to potentially sensitive dimensions such as SQL text	Supported	Supported	Yes	Yes	Instance
Use the Load Distribution component to analyze load distribution across instances within the same shard group	Supported	Supported	Yes	No	Instance
Analyze operating system processes happening in your databases with detailed metrics per running process Amazon RDS Enhanced Monitoring is required for this feature to work.	Not supported	Supported	No	Yes	Instance
Create and save fleet-wide monitoring views to assess health across hundreds of databases	Not supported	Supported	Yes	No	Fleet
Analyze SQL locks with 15 months of retention and a guided UX	Not supported	Not supported	No	No	Instance

Feature	Standard	Advanced	ShardGroup	Instance	Database Insights dashboard
Analyze SQL execution plans with 15 months of retention and guided UX	Not supported	Not supported	No	No	Instance
Visualize per-query statistics	Not supported	Supported	No	Yes	Instance
Analyze slow SQL queries Export of database logs to CloudWatch Logs is required for this feature to work.	Not supported	Supported	No	Yes	Instance
View calling services with CloudWatch Application Signals	Not supported	Supported	Yes	No	Both
View a consolidated dashboard for all database telemetry, including metrics, logs, events, and applications Export of database logs to CloudWatch Logs is required to view database logs in the Database Insights console.	Not supported	Supported	No	Yes	Instance

Feature	Standard	Advanced	ShardGroup	Instance	Database Insights dashboard
Import Performance Insights counter metrics into CloudWatch automatically	Not supported	Supported	N/A	N/A	Instance
View Amazon RDS events in CloudWatch	Not supported	Supported	Yes	No	Both
Analyze database performance for a time period of your choice with on-demand analysis	Not supported	Not supported	No	No	Instance

Note

Enhanced Monitoring is automatically enabled for Aurora PostgreSQL Limitless Databases. Enhanced Monitoring incurs additional charges. For more information, see Cost of Enhanced Monitoring.

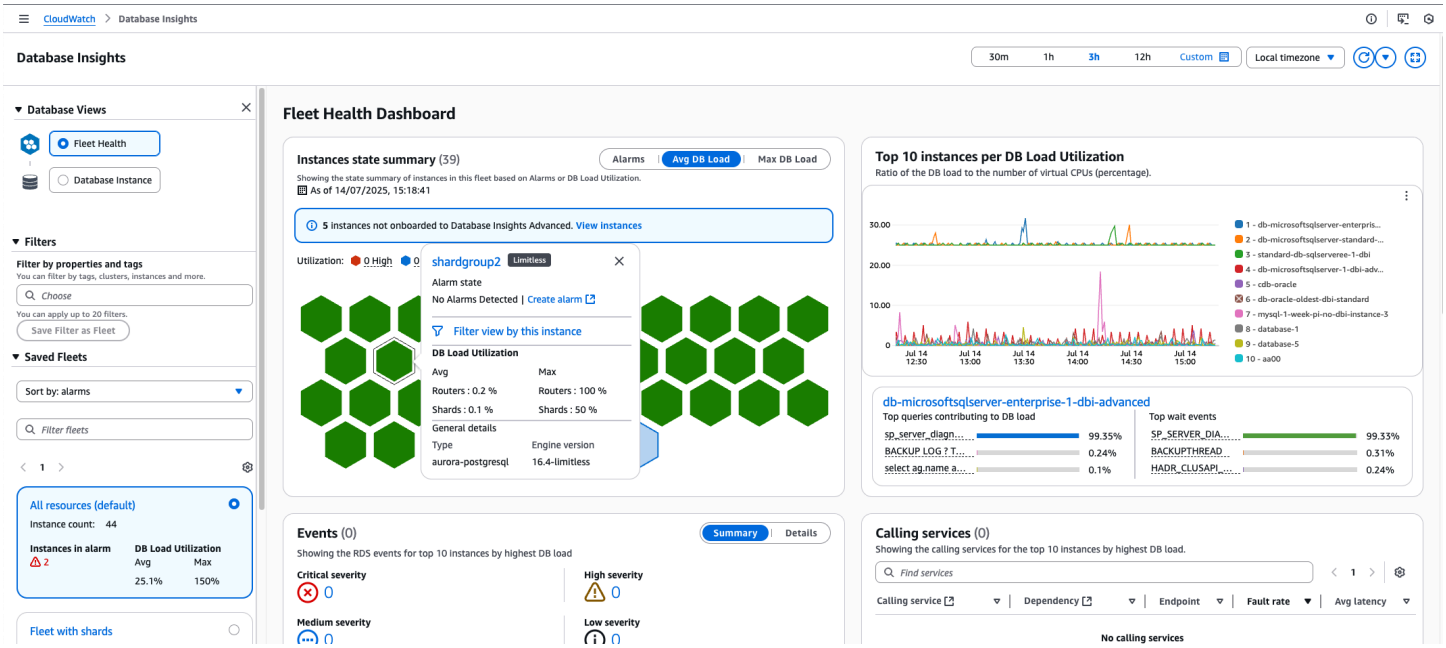
For Aurora PostgreSQL Limitless Databases, logs are automatically published to CloudWatch Logs and are discoverable in the Database Insights console. This incurs additional charges, following standard CloudWatch Logs pricing. For details about how CloudWatch Logs and Database Insights are priced and pricing examples, see Amazon CloudWatch pricing.

Monitoring Aurora Limitless shard groups in the Fleet Health Dashboard

Database Insights supports monitoring Aurora Limitless shard groups in the Fleet Health Dashboard.

In this view, you can see your Limitless shard groups alongside other databases that make up your database fleets. The Fleet Health Dashboard provides an opinionated view of health and

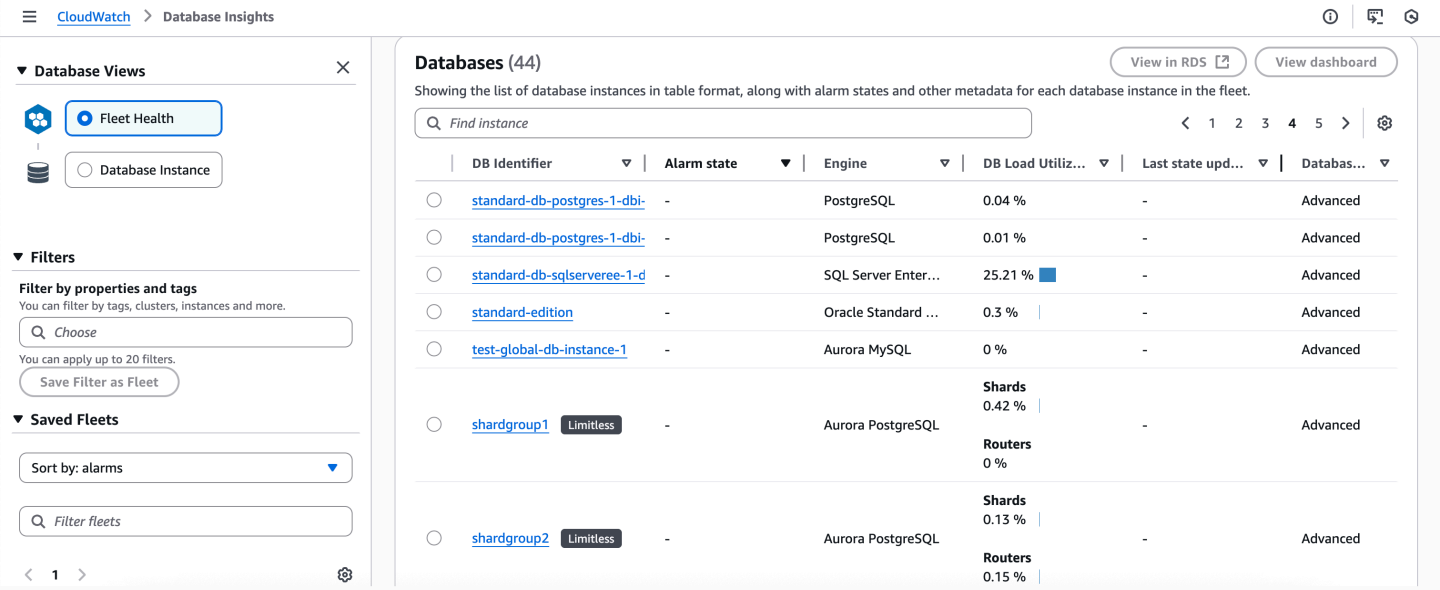
DBLoad utilization for your Limitless shard groups, similar to how it presents information for other databases in the fleet.



When viewing Aurora Limitless databases in the Fleet Health Dashboard:

- Only shard groups are visible, not individual instances
- Shard groups appear in the following widgets:
  - The honeycomb chart
  - The top 10 by DBLoad
  - Events
  - Calling Services
  - The table list
- DBLoad utilization is provided for both routers and shards

This fleet-level view allows you to monitor and compare the performance of your Aurora Limitless shard groups with other databases in your fleet, providing a comprehensive overview of your entire database fleet.



Monitoring Aurora PostgreSQL Limitless Databases in the Instance Dashboard

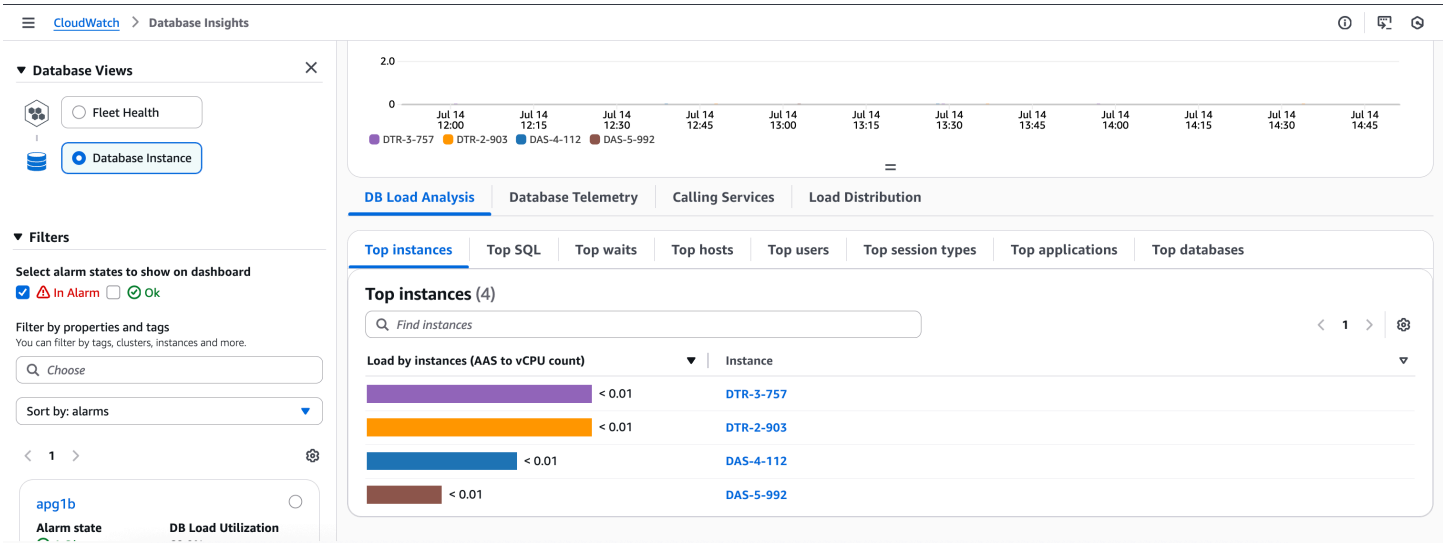
Database Insights works similarly for Aurora PostgreSQL Limitless Database as it does for standard Aurora DB clusters. However, you track metrics at the shard group level for Aurora PostgreSQL Limitless Database. The two main metrics to track are the following:

- **Database load** – Measures the level of activity in your database. The key metric is `DBLoad`, which is collected every second. The unit for the `DBLoad` metric is average active sessions (AAS). To get the average active sessions, Database Insights samples the number of sessions concurrently running a query. The AAS is the total number of sessions divided by the total number of samples for a specific time period.
- **Maximum CPU** – The maximum computational power available to your database. To see whether active sessions are exceeding the maximum CPU, look at their relationship to the `Max vCPU` line. The `Max vCPU` value is determined by the number of vCPU (virtual CPU) cores for your DB instance.

You can also "slice" the `DBLoad` metric into dimensions, which are subcategories of the metric. The most useful dimensions are the following:

- **Top instances** – Shows the relative DB load for your instances (shards and routers) in descending order.
- **Wait events** – Cause SQL statements to wait for specific events to happen before they can continue running. Wait events indicate where work is impeded.

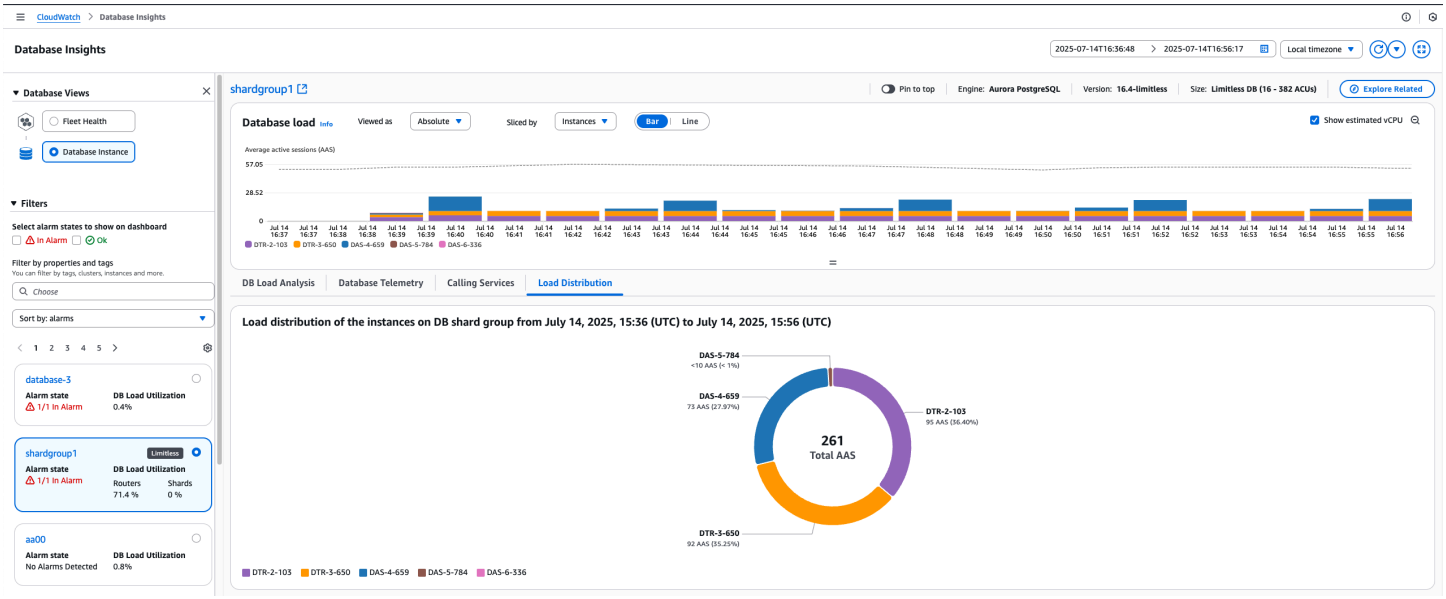
- **Top SQL** – Shows which queries contribute the most to DB load.



Analyze DB load for Aurora PostgreSQL Limitless Databases with Database Insights

With Database Insights, you can track metrics at the shard group level and at the instance level for an Aurora PostgreSQL Limitless Database. When analyzing DB load for an Aurora PostgreSQL Limitless Database, you might want to compare the DB load for each shard and router to the maximum vCPU.

The Absolute view shows the number of Average active sessions (AAS) and the estimated vCPU. The Relative view shows the ratio of AAS to the estimated vCPU.



Analyzing relative DB load using the Database Insights dashboard

You might want to improve the performance of your Aurora PostgreSQL Limitless Database by tracking relative DB load. To analyze relative DB load by instance for your Aurora PostgreSQL Limitless Database, use the following procedure.

To analyze relative DB load using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose Database Insights.
3. Choose an Aurora PostgreSQL Limitless Database. The Database Insights dashboard is displayed for that Aurora PostgreSQL Limitless Database.
4. In the Database load (DB load) section, choose *Instances* for *Sliced by*. To see the ratio of Average active sessions (AAS) to vCPU cores for all of the instances in your Aurora PostgreSQL Limitless Database, choose *Relative* for *Viewed as*.

   The Average active sessions chart shows the DB load for instances in yourAurora PostgreSQL Limitless Database.
5. To view the top instances, choose the *Top instances* tab.
6. (Optional) To analyze DB load for an instance in your Aurora PostgreSQL Limitless Database, choose the instance name in the *Instances* column.

Analyzing DB load by waits using the Database Insights dashboard

You might want to improve the performance for your Aurora PostgreSQL Limitless Database by tracking wait events. To analyze DB load by wait events for your Aurora PostgreSQL Limitless Database, use the following procedure.

To analyze DB load by waits for Aurora PostgreSQL Limitless Database using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose Database Insights.
3. Choose an Aurora PostgreSQL Limitless Database. The Database Insights dashboard is displayed for that Aurora PostgreSQL Limitless Database.
4. In the Database load (DB load) section, choose *Waits* for *Sliced by*. To view the number of AAS and the estimated vCPU, choose *Absolute* for *Viewed as*.

The Average active sessions chart shows the DB load for instances in your Aurora PostgreSQL Limitless Database.

5. Scroll down to the *Top SQL* tab.
6. Choose the SQL statement to expand it into its component statements.

Analyzing load distribution using the Database Insights dashboard

You might want to balance the load distribution for instances on your Aurora PostgreSQL Limitless Database. To analyze load distribution of the instances on an Aurora PostgreSQL Limitless Database, use the following procedure.

To analyze load distribution of the instances on an Aurora PostgreSQL Limitless Database using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose Database Insights.
3. Choose an Aurora PostgreSQL Limitless Database. The Database Insights dashboard is displayed for that Aurora PostgreSQL Limitless Database.
4. In the Database load (DB load) section, choose *Instances* for *Sliced by*. To view the number of AAS and the estimated vCPU for all instances in your Aurora PostgreSQL Limitless Database, choose *Absolute* for *Viewed as*.

   The Average active sessions chart shows the DB load for instances in your Aurora PostgreSQL Limitless Database.
5. To see a chart of the load distribution of the instances in your Aurora PostgreSQL Limitless Database, choose the *Load distribution* tab.

Troubleshooting for CloudWatch Database Insights

Use the following information to troubleshoot issues for CloudWatch Database Insights.

Applying tags to Amazon RDS resources

To apply tags to your databases, use the Amazon RDS API, AWS CLI, or Amazon RDS console. For more information, see the following topics.

- AddTagsToResource in the *Amazon RDS API Reference*

- add-tags-to-resource in the *Amazon RDS Command Line Reference*
- Tagging Amazon Aurora and Amazon RDS resources in the *Amazon Aurora User Guide*

Maximum DB instances for fleets

You can't monitor more than 500 DB instances in a database fleet. You can use filters to create a fleet health view with less than 500 DB instances.

SageMaker AI Insights

Use SageMaker AI Insights to monitor and troubleshoot SageMaker AI inference endpoints at scale. The dashboard displays curated metrics and visualizations across three views—**Performance**, **Capacity**, and **Reliability**—so you can quickly identify issues, optimize resource utilization, and make sure high availability across your endpoints.

SageMaker AI Insights supports monitoring across endpoint types (single-model endpoints and inference component (IC)-based endpoints) and inference frameworks (vLLM, SGLang, TensorRT-LLM).

To get started with SageMaker AI Insights, see the following topics.

Topics

- Get started with CloudWatch SageMaker AI Insights
- SageMaker AI Insights dashboard
- SageMaker AI Insights OpenTelemetry metrics reference
- Troubleshooting SageMaker AI Insights

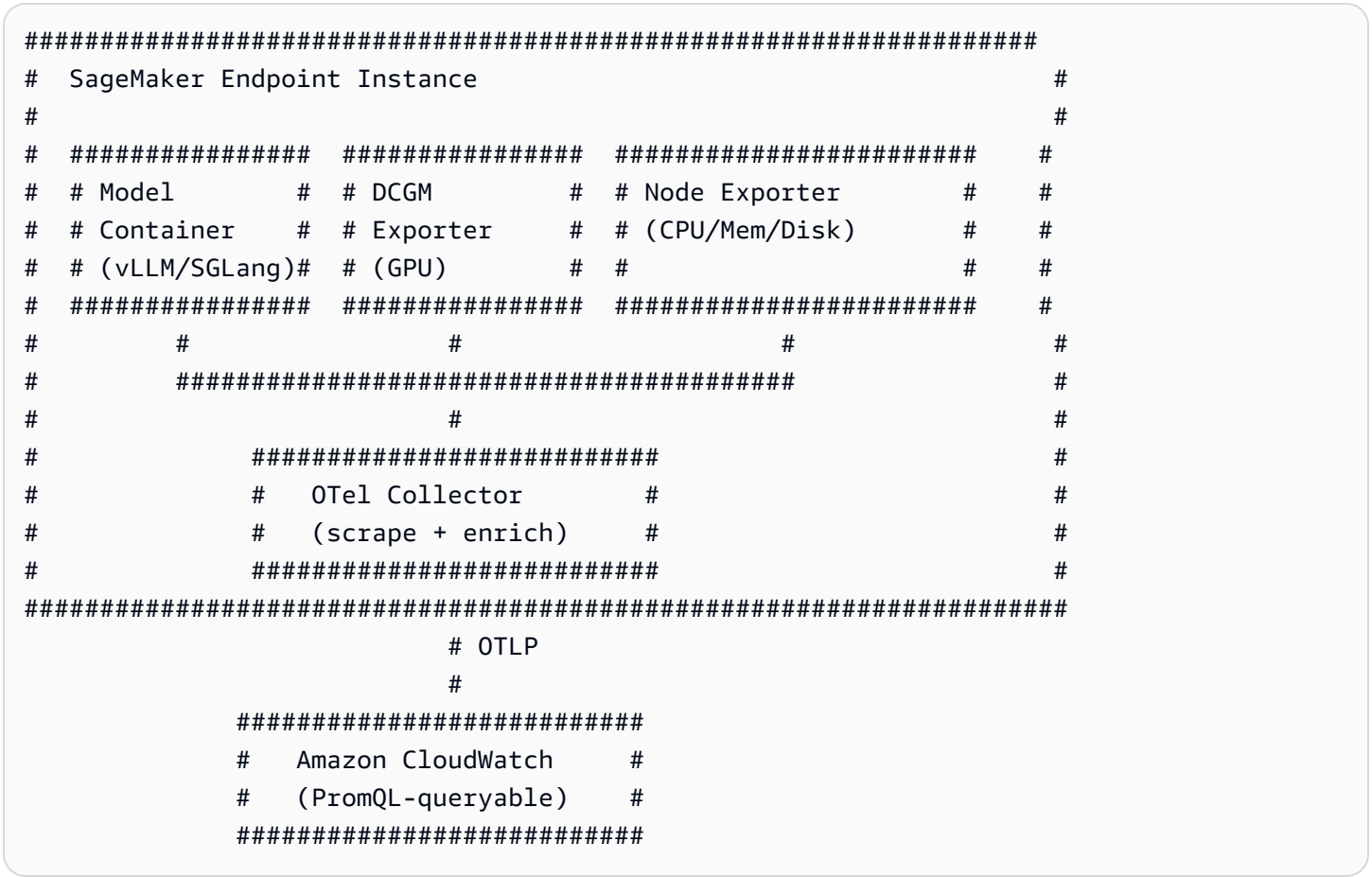
Key capabilities

- **OpenTelemetry-native collection.** Metrics are collected using an OTel Collector that scrapes Prometheus endpoints from DCGM (GPU metrics), node exporters (CPU, memory, disk), and inference framework containers (vLLM, SGLang).
- **Rich dimensional labels.** Every metric includes labels such as `aws.sagemaker.endpoint.name`, `aws.sagemaker.inference_component.name`, `@resource.host.id`, `@resource.cloud.availability_zone`, and `@resource.host.type`.
- **Per-GPU attribution.** GPU metrics (DCGM) include per-inference-component attribution for multi-tenant instances.
- **Inference framework metrics.** Native vLLM and SGLang metrics—tokens per second, time to first token (TTFT), inter-token latency, KV cache utilization, queue depth, batch size—without custom instrumentation.
- **PromQL query support.** Query in CloudWatch, CloudWatch Query Studio, or Amazon Managed Grafana.

- **Configurable scrape frequency.** Set through `MetricPublishFrequencyInSeconds` (10, 30, 60, 120, 180, 240, or 300 seconds; default 60). Control plane metrics are event-driven.

Architecture and data flow

1. The **model container**, **DCGM exporter**, and **node exporter** expose Prometheus-compatible metrics on the instance.
2. The **OTel Collector** scrapes these endpoints and enriches each metric with labels (endpoint name, IC name, instance ID, AZ).
3. Enriched metrics are **exported through OTLP** to CloudWatch.
4. Metrics are **queryable through PromQL** in CloudWatch.



How to access

You can open SageMaker AI Insights from the SageMaker AI console by using any of the following paths.

Source	Action	Destination
SageMaker AI console → **Endpoints** list	Choose **Open SageMaker AI Insights**	Fleet-level dashboard (no filter)
SageMaker AI console → endpoint detail page	Choose **View in SageMaker AI Insights**	Dashboard filtered to that endpoint
SageMaker AI console → **IC** tab → per-IC row	Choose **Metrics**	Dashboard filtered to endpoint and inference component

You can also navigate directly through the CloudWatch console by choosing **Infrastructure monitoring** → **SageMaker AI Insights**.

Prerequisites

- At least one SageMaker AI inference endpoint in the `InService` status
- `EnableDetailedObservability` set to `true` on the endpoint configuration (default for new endpoints)
- OTel enrichment enabled in your account
- The `cloudwatch:GetMetricData` and `cloudwatch:ListMetrics` IAM permissions

For more information, see Get started with CloudWatch SageMaker AI Insights.

Pricing

CloudWatch OpenTelemetry metric ingestion applies. For more information, see Amazon CloudWatch Pricing.

Get started with CloudWatch SageMaker AI Insights

Follow these steps to enable and access SageMaker AI Insights for your inference endpoints.

For the full setup guide on enabling detailed observability metrics on your endpoints, see SageMaker AI detailed observability in the *SageMaker AI Developer Guide*.

Step 1: Enable OTel enrichment

SageMaker AI Insights requires OTel metric enrichment and resource tags to be enabled in your account.

To enable OTel enrichment (console)

1. Open the CloudWatch console and choose **Settings**.
2. Enable **OTel metric enrichment**.
3. Enable **Resource tags for telemetry**.

To enable OTel enrichment with the AWS CLI, use the following commands.

```
aws cloudwatch start-otel-enrichment
aws observabilityadmin start-telemetry-enrichment
```

Step 2: Verify your endpoint configuration

Make sure that `EnableDetailedObservability` is set to `true` on your endpoint configuration. This is the default for new endpoints.

```
aws sagemaker describe-endpoint-config --endpoint-config-name name
# Look for "EnableDetailedObservability": true
```

If it is not enabled, create a new endpoint configuration with the flag set to `true` and update your endpoint.

Step 3: Confirm your endpoint is InService

Confirm that your endpoint is in the `InService` state.

```
aws sagemaker describe-endpoint --endpoint-name name --query "EndpointStatus"
```

The expected output is `"InService"`.

Step 4: Open SageMaker AI Insights

Navigate to the dashboard by using any of the following paths.

- CloudWatch console → **Infrastructure monitoring** → **SageMaker AI Insights**
- SageMaker AI console → **Endpoints** → choose **Open SageMaker AI Insights**
- SageMaker AI console → endpoint detail page → choose **View in SageMaker AI Insights**

Step 5: Verify data is flowing

After 2–3 minutes, the dashboard should populate with the following data.

- Invocation counts in the summary bar
- Instance and inference component (IC) data in the **Performance** tab
- Utilization metrics in the **Capacity** tab

> **Tip**
>
> If you see **No data**, verify that OTel enrichment is enabled and your endpoint has been `InService` for at least 2–3 minutes.

Required permissions for SageMaker AI Insights

Certain IAM permissions are required to use SageMaker AI Insights. The following policy contains the permissions required to use SageMaker AI Insights.

```
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:ListEndpoints",
        "sagemaker:ListInferenceComponents"
    ],
    "Resource": "*"
}
```

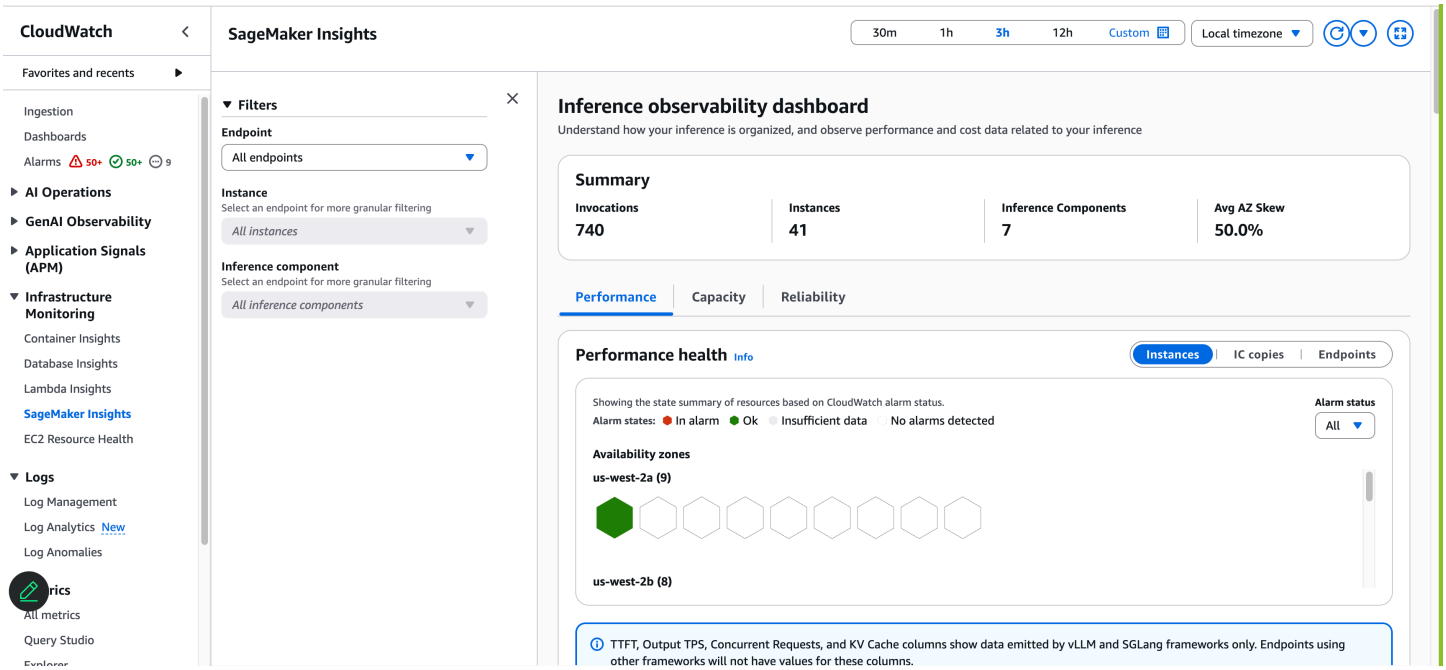
SageMaker AI Insights dashboard

The SageMaker AI Insights dashboard is organized into three tabs: **Performance**, **Capacity**, and **Reliability**. Each tab provides a focused view of your inference endpoint health.

Dashboard layout

Summary bar

The summary bar at the top of the dashboard shows fleet-wide totals.



Metric	Description
Invocations	Total invocations across all endpoints in the selected time range
Instances	Total number of instances currently serving traffic
Inference Components	Total number of inference components across all endpoints
Avg AZ Skew	Average availability zone distribution imbalance (0% = perfectly balanced)

Filters panel

Filter	Description
Endpoint	Select a specific endpoint or **All endpoints** for the fleet view
Instance	Select a specific instance (requires an endpoint selected first)
Inference component	Select a specific IC for granular filtering

Drill-down

SageMaker AI Insights supports progressive drill-down.

1. **Fleet level** (default)—all endpoints visible, summary metrics aggregated
2. **Endpoint level**—select an endpoint from the filter panel or choose an endpoint link in any table
3. **IC level**—select an inference component from the filter panel

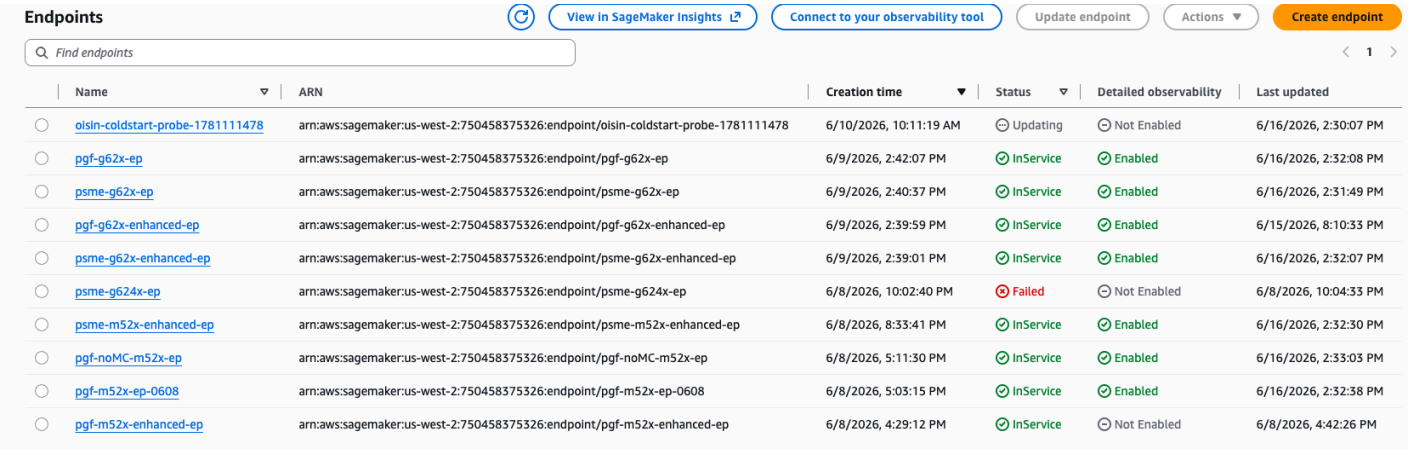
Cross-linking with the SageMaker AI console

- Choose an endpoint name to open the endpoint detail page in the SageMaker AI console.
- Choose **View logs** to open CloudWatch Logs filtered to that endpoint or IC.
- From the SageMaker AI console, choose **View in SageMaker AI Insights** or **Metrics** for a per-IC row.

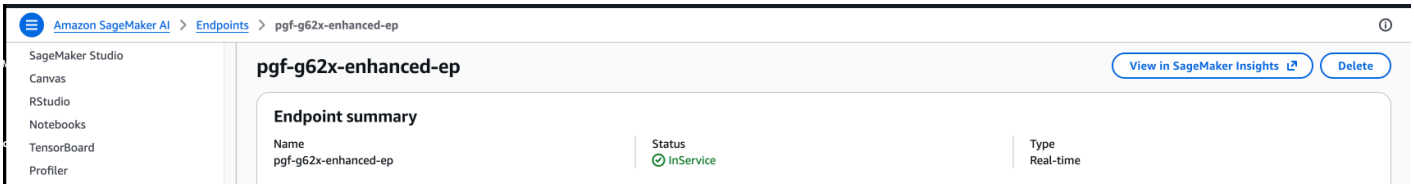
Accessing the dashboard

You can access the SageMaker AI Insights dashboard from multiple locations in the console.

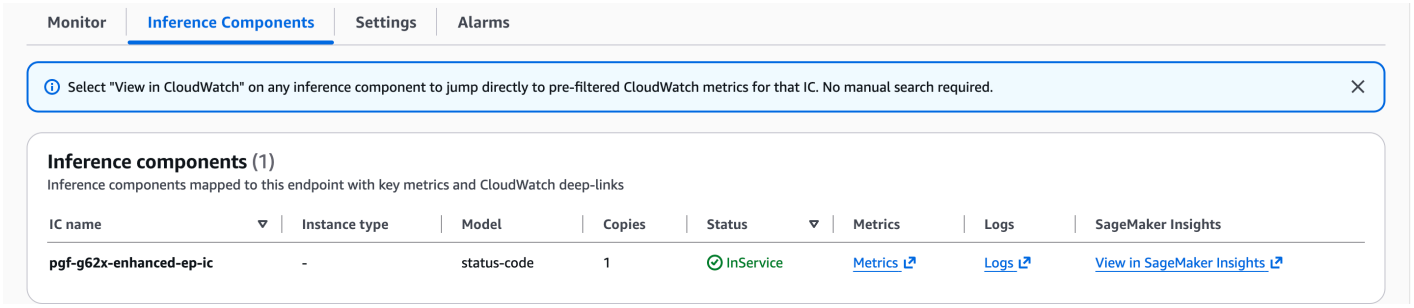
- **From the Endpoints list page:** Choose **View in SageMaker AI Insights** to open the dashboard at the fleet level with no filters applied.



- **From the endpoint detail page:** Choose **View in SageMaker AI Insights** to open the dashboard filtered to that endpoint.



- **From the inference component detail page:** Choose **View in SageMaker AI Insights** to open the dashboard filtered to that endpoint and inference component.



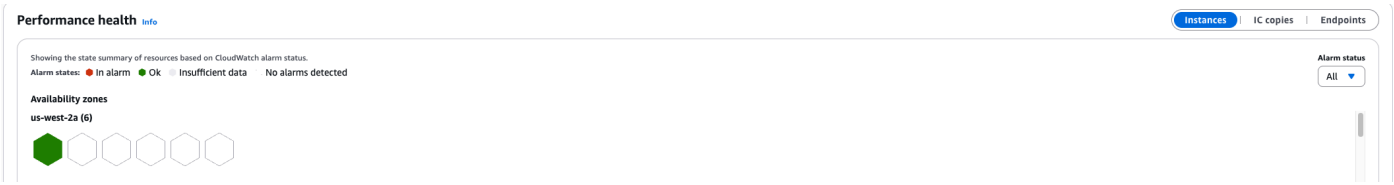
- **Direct navigation:** CloudWatch console → **Infrastructure monitoring** → **SageMaker AI Insights**.

Performance tab

The **Performance** tab answers "is it healthy?" and "why is it slow?"—flowing from fleet-wide health at the top to per-IC diagnostics at the bottom.

Performance health

Honeycombs grouped by AZ showing alarm state for instances, IC copies, and endpoints at a glance.



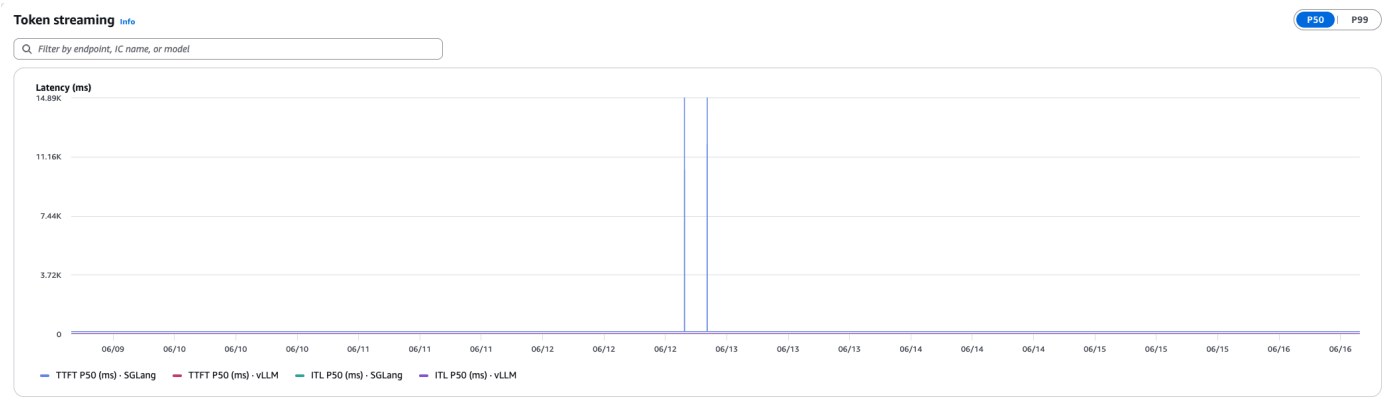
Instance performance table

Per-instance breakdown showing TTFT (P50/P99), output TPS (avg/max), concurrent requests (live/max), and KV cache utilization.



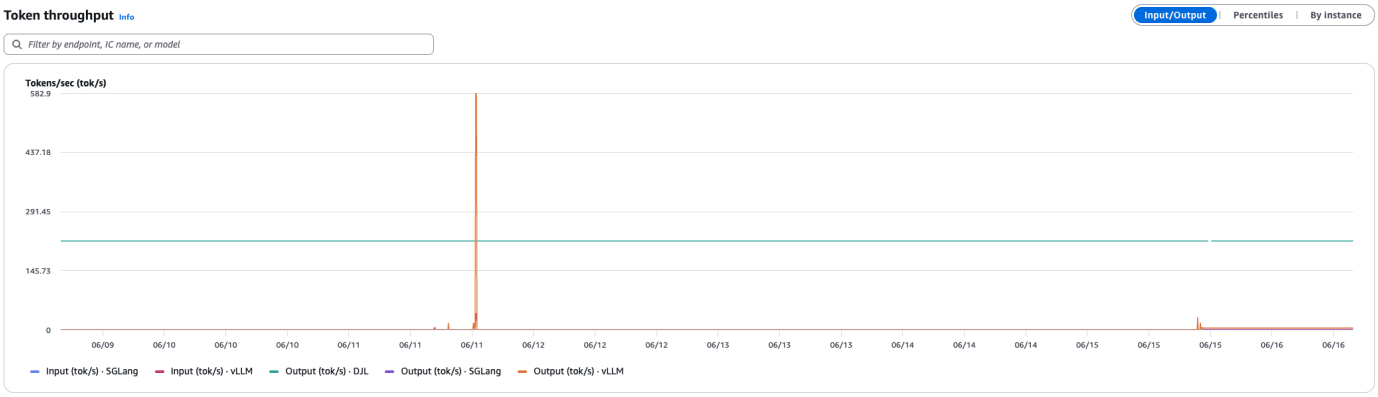
Token streaming

Time-series chart showing TTFT and inter-token latency (ITL) with a P50/P99 toggle, broken down by framework.



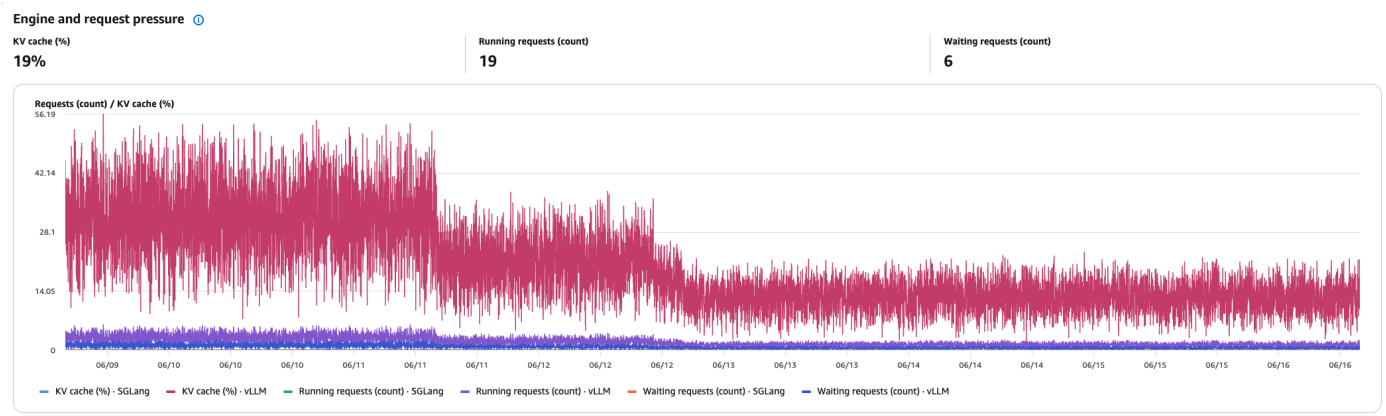
Token throughput

Input and output tokens per second by framework, with toggles for Input/Output, Percentiles, and By instance views.



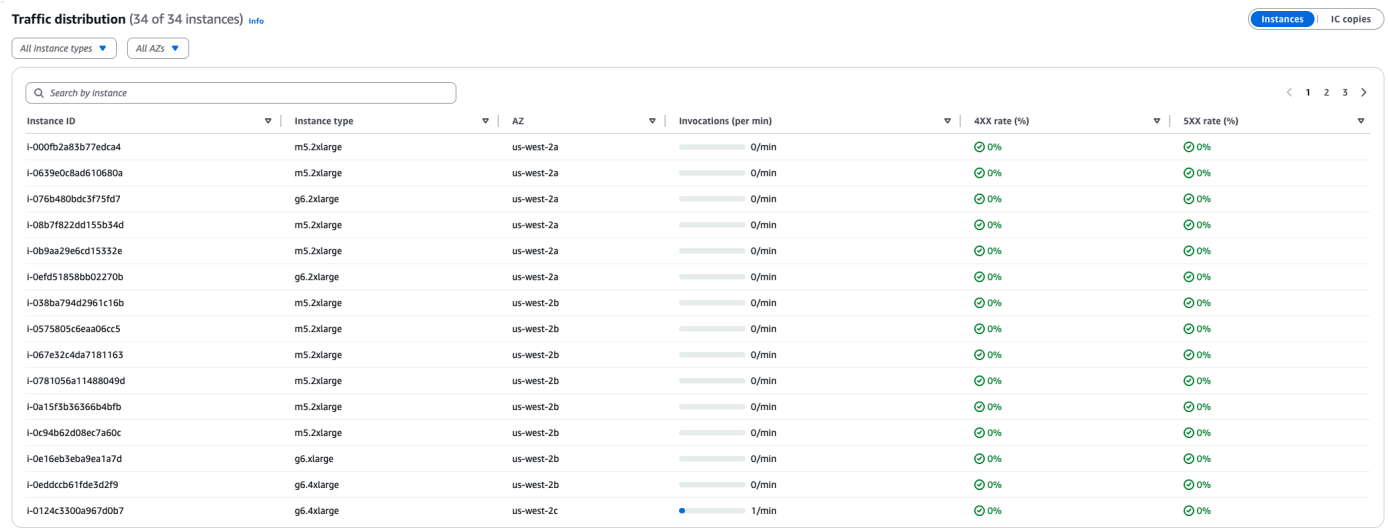
Engine and request pressure

KV cache utilization (%), running requests, and waiting requests over time—key saturation signals for the inference engine.



Traffic distribution

Per-instance or per-IC-copy table showing invocations per minute, 4XX rate, and 5XX rate to identify routing imbalances.



Error mix over time

Line chart showing 4XX, 5XX, and mid-stream error rates over time per IC.



Latency breakdown over time

Stacked area chart with tabs for Invoke (model latency + overhead latency) and Streaming (first chunk model + first chunk overhead), with a P50/P90 toggle.

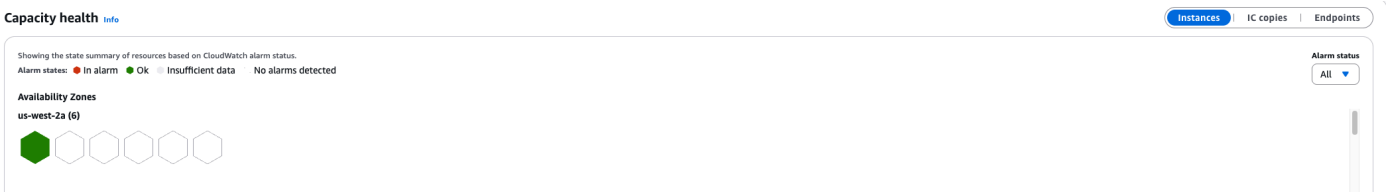


Capacity tab

The **Capacity** tab answers "do I have headroom?" and "is my hardware healthy?"—showing actual utilization compared to reserved capacity.

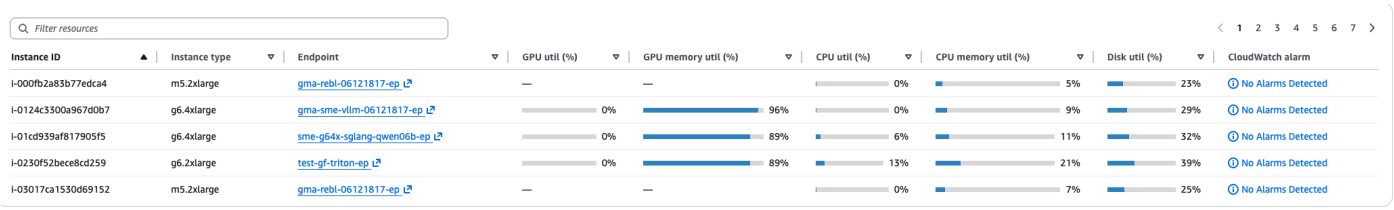
Capacity health

The same honeycomb visualization as the **Performance** tab, showing alarm state for instances, IC copies, and endpoints.



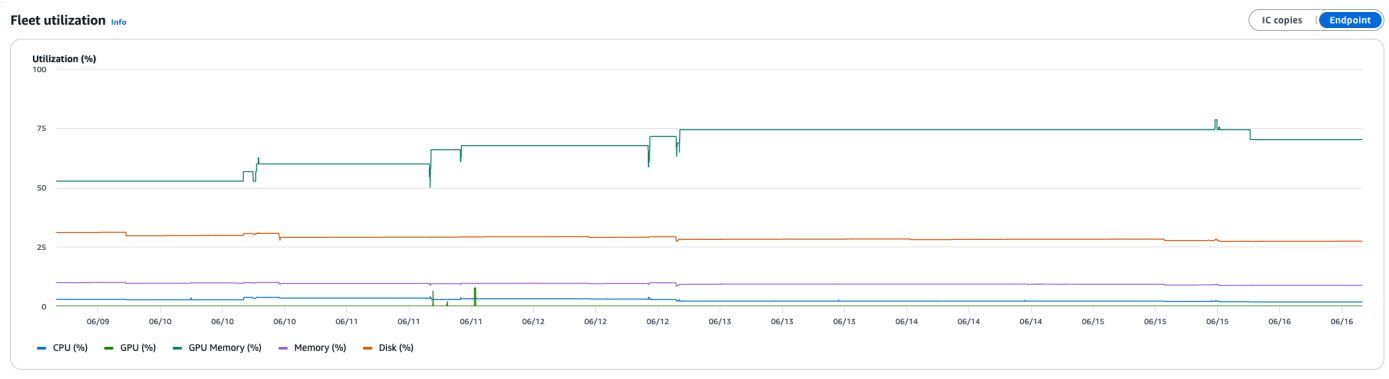
Instance capacity table

Per-instance utilization bars for GPU, GPU memory, CPU, memory, and disk.



Fleet utilization

Time-series showing CPU, GPU, GPU memory, memory, and disk utilization per instance, with toggles for Instance, IC copies, and Endpoint views.

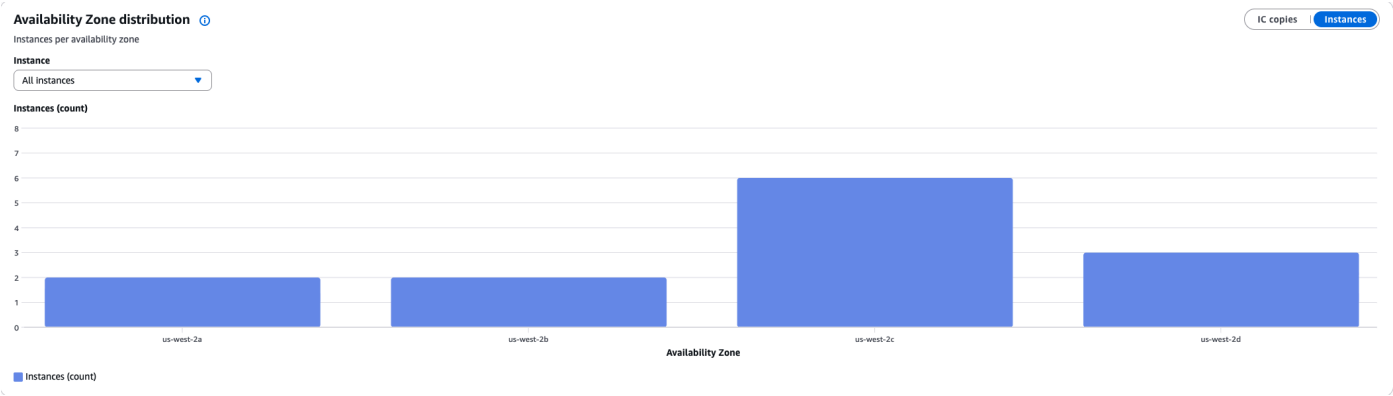


Reliability tab

The **Reliability** tab answers "is it resilient?" and "why did scaling fail?"—covering AZ distribution, scaling behavior, and provisioning events.

Availability Zone distribution

Bar chart showing instance or IC copy count per AZ to validate high availability (HA) compliance.



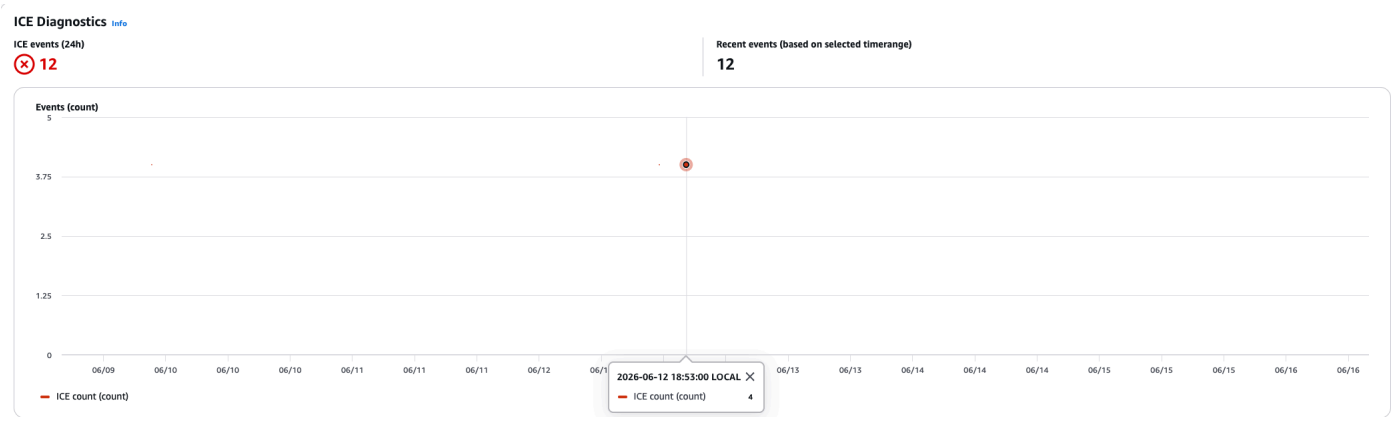
Cold start anatomy

Horizontal stacked bar showing the breakdown of provisioning time into model download, GPU load, container start, and platform overhead (IC endpoints only).



ICE diagnostics

Insufficient Capacity Error (ICE) count over time with an event table showing time, endpoint, failed instance type, and failed AZ. Non-zero values indicate capacity constraints.



SageMaker AI Insights OpenTelemetry metrics reference

This section provides a comprehensive list of OpenTelemetry metrics emitted by SageMaker AI detailed observability. Detailed observability is built on OpenTelemetry (OTel) and collects fine-grained operational metrics from the GPU, node, and inference framework layers, publishing them to CloudWatch with rich labels.

> **Note**
>
> Detailed observability publishes OpenTelemetry (OTel) metrics to CloudWatch through OTLP. These are not Prometheus metrics. The metrics are natively stored in CloudWatch as OTel metric data and are queryable using PromQL syntax.

Account-level aggregate metrics

Metric	Type	Description
`TotalEndpoints`	Aggregate	Count of all endpoints in the account
`TotalICs`	Aggregate	Count of all inference components
`TotalGPUs`	Aggregate	Count of all GPUs (derived from `DCGM_FI_DEV_GPU_UTIL`)

Endpoint-level metrics

Metric	Type	Description
InstanceCountPerEndpoint	Aggregate	Instance count serving traffic
InstancesPerAZ	Aggregate	Instances per availability zone
EmptyInstanceCount	Aggregate	Instances with no ICs placed
ScalingAction	Raw	Scaling event (in/out)
ICECount	Raw	Insufficient Capacity Error count
CPUUtilization (EP)	Aggregate	Endpoint-level CPU utilization
MemoryUtilization (EP)	Aggregate	Endpoint-level memory utilization
DiskUtilization (EP)	Aggregate	Endpoint-level disk utilization
E2EScalingLatency	Raw	End-to-end scaling duration
RebalancingType	Raw	Type of rebalancing event
RebalancingDuration	Raw	Rebalancing duration (seconds)
RebalancingCopiesMoved	Raw	IC copies moved during rebalancing
AZSkewDelta	Raw	AZ distribution imbalance
InstancesReleased	Raw	Instances freed by rebalancing

Inference framework metrics (vLLM / SGLang)

Metric	Type	Description
InputTokensPerSecond	Aggregate	Input token rate

Metric	Type	Description
OutputTokensPerSecond	Aggregate	Output token rate
TotalTPS	Aggregate	Total tokens per second
TPSUtilization	Aggregate	TPS utilization percentage
TTFT	Raw	Time to First Token (histogram)
InterTokenLatency	Raw	Inter-Token Latency (histogram)
KVCacheUtilization	Raw	KV cache usage percentage
QueueDepth	Raw	Waiting requests (queue)
BatchSize	Raw	Running requests (in-flight)
TPSPerInstance	Aggregate	TPS aggregated per instance
ConcurrentReqsPerCopy	Raw	Per-copy in-flight requests
FirstChunkLatencyPerIC	Raw	TTFT per IC
4XXErrorRatePerIC	Aggregate	Client errors per IC
5XXErrorRatePerIC	Aggregate	Server errors per IC
TTFTPerIC	Raw	TTFT filtered by IC
ITLPerIC	Raw	ITL filtered by IC
InputTPSPerIC	Aggregate	Input TPS per IC
OutputTPSPerIC	Aggregate	Output TPS per IC
KVCachePerIC	Raw	KV cache per IC

Metric	Type	Description
ModelErrorBreakdown	Aggregate	Error breakdown by type

> **Note**
>
> TTFT and InterTokenLatency are framework-dependent and only supported for vLLM and SGLang.

GPU (DCGM) metrics

These metrics are collected from the NVIDIA Data Center GPU Manager (DCGM) exporter. They are available on all GPU endpoints regardless of inference framework.

Metric	Type	Description
DCGM_FI_DEV_GPU_UTIL	Raw	GPU utilization (%)
DCGM_FI_DEV_MEM_COPY_UTIL	Raw	Memory copy utilization (%)
DCGM_FI_DEV_GPU_TEMP	Raw	GPU temperature (°C)
DCGM_FI_DEV_MEMORY_TEMP	Raw	Memory temperature (°C)
DCGM_FI_DEV_FB_FREE	Raw	Framebuffer memory free (bytes)
DCGM_FI_DEV_FB_USED	Raw	Framebuffer memory used (bytes)
DCGM_FI_DEV_SM_ACTIVE	Raw	Streaming multiprocessor active (%)

Node (instance) metrics

These metrics are collected from the node exporter. They provide instance-level visibility into CPU, memory, and disk utilization.

Metric	Type	Description
node_cpu_seconds_total	Raw	CPU time per mode per core (seconds)
node_memory_MemTotal_bytes	Raw	Total memory (bytes)
node_memory_MemAvailable_bytes	Raw	Available memory (bytes)
node_filesystem_size_bytes	Raw	Filesystem total size (bytes)
node_filesystem_avail_bytes	Raw	Filesystem available space (bytes)

HA and IC placement metrics

Metric	Type	Description
ICCopyCountPerAZ	Aggregate	IC copies per AZ
ICCopiesPerAZ	Aggregate	IC copies per AZ (alternative view)
AZSkewScore	Aggregate	AZ imbalance score
AZBalanceScore	Aggregate	Endpoint-level balance
ICDensityPerAZ	Aggregate	IC density per AZ
ICCopiesPerInstance	Aggregate	IC copies per instance
ICListPerInstance	Aggregate	Which ICs on which instance

Lifecycle and provisioning metrics

Metric	Type	Description
`ModelDownloadTime`	Raw	Time for Amazon S3 to instance download
`GPULoadTime`	Raw	Time for weights to GPU memory
`ContainerStartDuration`	Raw	Container start to health check
`ColdStartDuration`	Raw	End-to-end: creation to first invocation

Metrics available only as CloudWatch classic metrics

The following metrics are available only as CloudWatch classic metrics (namespace and dimension model) and require OTel enrichment to appear as OpenTelemetry metrics.

Metric	Notes
`InvocationsPerIC`	Blocked from direct OpenTelemetry emission
`InvocationsPerCopy`	Blocked
`ModelLatencyPerIC`	Blocked—use `vllm:e2e_request_latency_seconds` as the OTel alternative
`OverheadLatencyPerIC`	Blocked
`MidStreamErrorsPerIC`	Blocked (bidirectional streaming only)

Troubleshooting SageMaker AI Insights

The following sections describe common issues and resolutions for the CloudWatch SageMaker AI Insights dashboard. If your issue is not listed here, verify that your endpoint is `InService` and that OTel enrichment is enabled in your account.

Dashboard shows "No data" for all panels

Cause: OTel enrichment is not enabled in your account.

Resolution:

1. Open the CloudWatch console and choose **Settings**.
2. Enable **OTel metric enrichment**.
3. Enable **Resource tags for telemetry**.
4. Wait 2–3 minutes for metrics to begin flowing.

Alternatively, enable OTel enrichment with the following AWS CLI commands.

```
aws cloudwatch start-otel-enrichment
aws observabilityadmin start-telemetry-enrichment
```

TTFT, TPS, and KV cache columns show a dash

Cause: These metrics are only available for endpoints using the vLLM or SGLang frameworks. Endpoints using other frameworks (TensorRT-LLM, custom containers) do not emit these metrics.

Resolution: No action needed—this is expected behavior.

SageMaker AI Insights redirects to Database Insights

Cause: Known issue in some accounts.

Workaround: In the redirected URL, replace `instances` with `endpoints`. The dashboard then loads correctly.

Dashboard loads but shows 0 invocations

Cause: The endpoint is `InService` but not receiving traffic, or the selected time range has no invocations.

Resolution:

- Send test invocations to the endpoint.
- Adjust the time range to a period when traffic was active.

Metrics appear in CloudWatch but not in SageMaker AI Insights

Cause: `EnableDetailedObservability` might not be set to `true` on the endpoint configuration.

Resolution: Verify the endpoint configuration with the following command.

```
aws sagemaker describe-endpoint-config --endpoint-config-name name
# Look for "EnableDetailedObservability": true
```

Scaling events table is empty

Cause: Application Auto Scaling is not configured, or no scaling events occurred in the selected time range.

Resolution:

- Configure an autoscaling policy for your endpoint.
- Adjust the time range to include periods when scaling events occurred.

ElastiCache Insights

With ElastiCache Insights, you can monitor and troubleshoot your node-based Amazon ElastiCache for Valkey clusters. ElastiCache Insights is a prebuilt dashboard in the CloudWatch console. It shows the OpenTelemetry metrics from your cache engine and from the host that each node runs on.

For complete information about ElastiCache OpenTelemetry metrics, including supported configurations, metric attributes, example PromQL queries, and the metrics reference, see Monitoring ElastiCache with OpenTelemetry metrics in the *Amazon ElastiCache User Guide*.

Key capabilities

- **Vended metrics at no additional cost.** ElastiCache emits a core set of metrics automatically, at 60-second granularity, as soon as the cluster is running. You don't need to configure anything.
- **Detailed monitoring.** Select any metric with a resource metrics configuration to emit it at 15-second granularity. For more information, see CloudWatch detailed monitoring for OpenTelemetry metrics.
- **Rich attributes.** Every datapoint carries attributes that identify the account, Region, Availability Zone, replication group, shard, and node, so that you can filter and aggregate at query time.
- **PromQL query support.** Query metrics in Query Studio, create alarms from PromQL queries, add queries to dashboards, or query from Amazon Managed Grafana. For more information, see Query metrics with PromQL.
- **Customizable dashboard.** Use the prebuilt dashboard as is, or save a copy as a custom dashboard to add or remove widgets, or to combine ElastiCache metrics with metrics from other sources. Changes to a copy don't affect the prebuilt dashboard.

Supported configurations

All node-based Valkey clusters emit OpenTelemetry metrics, on any Valkey version, except clusters on AWS Outposts. Serverless caches, Redis OSS, and Memcached don't emit OpenTelemetry metrics.

How to access

To open the ElastiCache Insights dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Infrastructure monitoring**, **ElastiCache Insights**.

You can also open the dashboard from the **Metrics** tab of a cluster in the ElastiCache console.

Pricing

ElastiCache Insights is priced on CloudWatch OpenTelemetry metrics ingestion. The dashboard and the core set of vended metrics are free. When you select metrics for detailed monitoring, you pay the standard CloudWatch OpenTelemetry metrics ingestion price for the additional datapoints emitted. PromQL queries that run in the CloudWatch console are free. Queries through the CloudWatch API, including from Amazon Managed Grafana, and PromQL alarms are charged separately. If you save a copy of the dashboard, the copy is billed as a custom dashboard. For more information, see Amazon CloudWatch Pricing.

Get started with ElastiCache Insights

ElastiCache Insights requires no setup. As soon as a node-based Valkey cluster is running, ElastiCache emits a core set of vended OpenTelemetry metrics automatically, at 60-second granularity and at no additional cost, and the dashboard runs on these metrics. To open the dashboard, see How to access.

Prerequisites

- A node-based Amazon ElastiCache for Valkey cluster that emits OpenTelemetry metrics. For eligibility, see Supported configurations.
- The `cloudwatch:GetMetricData` and `cloudwatch:ListMetrics` IAM permissions to view metrics in the dashboard.

To emit additional metrics at 15-second granularity, select them with a resource metrics configuration. For more information, see CloudWatch detailed monitoring for OpenTelemetry metrics.

Use Contributor Insights to analyze high-cardinality data

You can use Contributor Insights to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. This helps you find top talkers and understand who or what is impacting system performance. For example, you can find bad hosts, identify the heaviest network users, or find the URLs that generate the most errors.

You can build your rules from scratch, and when you use the AWS Management Console you can also use sample rules that AWS has created. Rules define the log fields that you want to use to define contributors, such as `IpAddress`. You can also filter the log data to find and analyze the behavior of individual contributors.

CloudWatch also provides built-in rules that you can use to analyze metrics from other AWS services.

All rules analyze incoming data in real time.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules in that monitoring account that analyze log groups in source accounts and in the monitoring account. You can also create a single rule that analyzes log groups in multiple accounts. For more information, see CloudWatch cross-account observability.

With Contributor Insights, you are charged for each occurrence of a log event that matches a rule. For more information, see Amazon CloudWatch Pricing.

> **Note**
>
> Contributor Insights can only match log entries when the numeric values that the rule references are between -1e9 and 1e9. If a value in a log entry is outside of this range, Contributor Insights skips that log entry.

Topics

- Create a Contributor Insights rule in CloudWatch
- Contributor Insights rule syntax in CloudWatch
- CloudWatch Contributor Insights rule examples

- Viewing Contributor Insights reports in CloudWatch
- Graphing metrics generated by rules in CloudWatch
- Using Contributor Insights built-in rules in CloudWatch

Create a Contributor Insights rule in CloudWatch

You can create rules to analyze log data. Any logs in JSON or Common Log Format (CLF) can be evaluated. This includes your custom logs that follow one of these formats and logs from AWS services such as Amazon VPC flow logs, Amazon Route 53 DNS query logs, Amazon ECS container logs, and logs from AWS CloudTrail, Amazon SageMaker AI, Amazon RDS, AWS AppSync and API Gateway.

In a rule, when you specify field names or values, all matching is case sensitive.

You can use built-in sample rules when you create a rule or you can create your own rule from scratch. Contributor Insights includes sample rules for the following types of logs:

- Amazon API Gateway logs
- Amazon Route 53 public DNS query logs
- Amazon Route 53 resolver query logs
- CloudWatch Container Insights logs
- VPC flow logs

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can create Contributor Insights rules for log groups in the source accounts linked to this monitoring account, in addition to creating rules for log groups in the monitoring account. You can also set up a single rule that monitors log groups in different accounts. For more information, see CloudWatch cross-account observability.

> **Important**
>
> When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see Condition keys for Contributor Insights log group access.

To create a rule using a built-in sample rule

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts. You can also set up a single rule to analyze log groups in different accounts.

   - (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix. If this is a monitoring account, you can optionally select the accounts to search in, otherwise all accounts are selected.

   > **Note**
   >
   > You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

5. For **Rule type**, choose **Sample rule**. Then choose **Select sample rule** and select the rule.
6. For **Log format**, select the format of the logs that the rule will evaluate.
7. For **Transformers**, select **Enable contributor insights on transformers** to have the rule evaluate log events after they are transformed by CloudWatch Logs log transformation. If you select to enable this:

   - If the rule evaluates any log groups that have transformers, the rule is applied to the transformed versions of the logs.
   - If the rule evaluates any log groups that don't have transformers, the rule is applied to the original versions of the logs.

   If you don't select this, the evaluation happens to original log events in all log groups, even those that use log transformation.

> **Note**
>
> If a log group has a transformer, and transformation fails for some log events, those log events won't be evaluated by Contributor Insights. For information about investigating log transformation failures, see Transformation metrics and errors.

8. The sample rule you chose has filled out the **Log format**, **Contribution**, **Filters**, and **Aggregation** fields. You can adjust those values, if you like.
9. Choose **Next**.
10. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, (hyphen), (underscore), and (period).
11. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see Amazon CloudWatch Pricing.

    Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.
12. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see Tagging your Amazon CloudWatch resources.
13. Choose **Create**.

To create a rule from scratch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. Choose **Create rule**.
4. For **Select log group(s)**, select the log group(s) that you want your rule to monitor. You can select as many as 20 log groups. If you are signed in to a monitoring account that is set up for CloudWatch cross-account observability, you can select log groups in source accounts. You can also set up a single rule to analyze log groups in different accounts.

   - (Optional) To select all log groups that have names beginning with a specific string, choose the **Select by prefix match** dropdown, and then enter the prefix.

> **Note**
>
> You incur charges for each log event that matches your rule. If you choose the **Select by prefix match** dropdown, be aware of how many log groups the prefix can match. If you search more log groups than you want, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

5. For **Rule type**, choose **Custom rule**.
6. For **Log format**, choose **JSON** or **CLF**.
7. For **Transformers**, select **Enable contributor insights on transformers** to have the rule evaluate log events after they are transformed by CloudWatch Logs log transformation. If you select to enable this:

   - If the rule evaluates any log groups that have transformers, the rule is applied to the transformed versions of the logs.
   - If the rule evaluates any log groups that don't have transformers, the rule is applied to the original versions of the logs.

   If you don't select this, the evaluation happens to original log events in all log groups, even those that use log transformation.

   > **Note**
   >
   > If a log group has a transformer, and transformation fails for some log events, those log events won't be evaluated by Contributor Insights. For information about investigating log transformation failures, see Transformation metrics and errors.

8. You can finish creating the rule by using the wizard or by choosing the **Syntax** tab and specifying your rule syntax manually.

   To continue using the wizard, do the following:

   a. For **Contribution**, **Key**, enter a contributor type that you want to report on. The report displays the top-N values for this contributor type.

Valid entries are any log field that has values. Examples include **`requestId`**, **`sourceIPaddress`**, and **`containerID`**.

For information about finding the log field names for the logs in a certain log group, see Finding Log Fields.

Keys larger than 1 KB are truncated to 1KB.

b. (Optional) Choose **Add new key** to add more keys. You can include as many as four keys in a rule. If you enter more than one key, the contributors in the report are defined by unique value combinations of the keys. For example, if you specify three keys, each unique combination of values for the three keys is counted as a unique contributor.

c. (Optional) If you want to add a filter that narrows the scope of your results, choose **Add filter**. For **Match**, enter the name of the log field that you want to filter on. For **Condition**, choose a comparison operator, and enter a value that you want to filter for.

You can add as many as four filters in a rule. Multiple filters are joined by AND logic, so only log events that match all filters are evaluated.

> **Note**
>
> Arrays that follow comparison operators, such as `In`, `NotIn`, or `StartsWith`, can include as many as 10 string values. For more information about the Contributor Insights rules syntax, see Contributor Insights rule syntax in CloudWatch.

d. For **Aggregate on**, choose **Count** or **Sum**. Choosing **Count** causes the contributor ranking to be based on the number of occurrences. Choosing **Sum** causes the ranking to be based on the aggregated sum of the values of the field that you specify for **Contribution**, **Value**.

9. To enter your rule as a JSON object instead of using the wizard, do the following:

a. Choose the **Syntax** tab.

b. In **Rule body**, enter the JSON object for your rule. For information about rule syntax, see Contributor Insights rule syntax in CloudWatch.

10. Choose **Next**.

11. For **Rule name**, enter a name. Valid characters are A-Z, a-z, 0-9, "-", "_', and ".".

12. Choose whether to create the rule in a disabled or enabled state. If you choose to enable it, the rule immediately starts analyzing your data. You incur costs when you run enabled rules. For more information, see Amazon CloudWatch Pricing.

    Contributor Insights analyzes only new log events after a rule is created. A rule cannot process logs events that were previously processed by CloudWatch Logs.
13. (Optional) For **Tags**, add one or more key-value pairs as tags for this rule. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see Tagging your Amazon CloudWatch resources.
14. Choose **Next**.
15. Confirm the settings that you entered, and choose **Create rule**.

You can disable, enable, or delete rules that you have created.

To enable, disable, or delete a rule in Contributor Insights

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. In the list of rules, select the check box next to a single rule.

   Built-in rules are created by AWS services and can't be edited, disabled, or deleted.
4. Choose **Actions**, and then choose the option you want.

Finding log fields

When you create a rule, you need to know the names of fields in the log entries in a log group.

To find the log fields in a log group

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, under **Logs**, choose **Insights**.
3. Above the query editor, select one or more log groups to query.

   When you select a log group, CloudWatch Logs Insights automatically detects fields in the data in the log group and displays them in the right pane in **Discovered fields**.

Contributor Insights rule syntax in CloudWatch

This section explains the syntax for Contributor Insights rules. Use this syntax only when you are creating a rule by entering a JSON block. If you use the wizard to create a rule, you don't need to know the syntax. For more information about creating rules using the wizard, see Create a Contributor Insights rule in CloudWatch.

All matching of rules to log event field names and values is case sensitive.

The following example illustrates the syntax for JSON logs.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*",
        "Log-group-name2"
    ],
    "LogFormat": "JSON",
    "Contribution": {
        "Keys": [
            "$.ip"
        ],
        "ValueOf": "$.requestBytes",
        "Filters": [
            {
                "Match": "$.httpMethod",
                "In": [
                    "PUT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Fields in Contributor Insights rules

Schema

The value of `Schema` for a rule that analyzes CloudWatch Logs data must always be `{"Name": "CloudWatchLogRule", "Version": 1}`

LogGroupNames

An array of strings. For each element in the array, you can optionally use `*` at the end of a string to include all log groups with names that start with that prefix.

Be careful about using wildcards with log group names. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

LogGroupARNs

If you are creating this rule in a CloudWatch cross-account observability monitoring account, you can use `LogGroupARNs` to specify log groups in source accounts that are linked to the monitoring account, and to specify log groups in the monitoring account itself. You must specify either `LogGroupNames` or `LogGroupARNs` in your rule, but not both.

`LogGroupARNs` is an array of strings. For each element in the array, you can optionally use `*` as a wildcard in certain situations. For example you can specify `arn:aws:logs:us-west-1:*:log-group/MyLogGroupName2` to specify log groups named `MyLogGroupName2` in all source accounts and in the monitoring account, in the US West (N. California) Region. You can also specify `arn:aws:logs:us-west-1:111122223333:log-group/GroupNamePrefix*` to specify all log groups in US West (N. California) in `111122223333` that have names starting with `GroupNamePrefix`.

You can't specify a partial AWS account ID as a prefix with a wild card.

Be careful about using wildcards with log group ARNs. You incur charges for each log event that matches a rule. If you accidentally search more log groups than you intend, you might incur unexpected charges. For more information, see Amazon CloudWatch Pricing.

LogFormat

Valid values are `JSON` and `CLF`.

Contribution

This object includes a `Keys` array with as many as four members, optionally a single `ValueOf`, and optionally an array of as many as four `Filters`.

Keys

An array of up to four log fields that are used as dimensions to classify contributors. If you enter more than one key, each unique combination of values for the keys is counted as a unique contributor. The fields must be specified using JSON property format notation.

ValueOf

(Optional) Specify this only when you are specifying `Sum` as the value of `AggregateOn`. `ValueOf` specifies a log field with numerical values. In this type of rule, the contributors are ranked by their sum of the value of this field, instead of their number of occurrences in the log entries. For example, if you want to sort contributors by their total `BytesSent` over a period, you would set `ValueOf` to `BytesSent` and specify `Sum` for `AggregateOn`.

Filters

Specifies an array of as many as four filters to narrow the log events that are included in the report. If you specify multiple filters, Contributor Insights evaluates them with a logical AND operator. You can use this to filter out irrelevant log events in your search or you can use it to select a single contributor to analyze their behavior.

Each member in the array must include a `Match` field and a field indicating the type of matching operator to use.

The `Match` field specifies a log field to evaluate in the filter. The log field is specified using JSON property format notation.

The matching operator field must be one of the following: `In`, `NotIn`, `StartsWith`, `GreaterThan`, `LessThan`, `EqualTo`, `NotEqualTo`, or `IsPresent`. If the operator field is `In`, `NotIn`, or `StartsWith`, it is followed by an array of string values to check for. Contributor Insights evaluates the array of string values with an OR operator. The array can include as many as 10 string values.

If the operator field is `GreaterThan`, `LessThan`, `EqualTo`, or `NotEqualTo`, it is followed by a single numerical value to compare with.

If the operator field is `IsPresent`, it is followed by either `true` or `false`. This operator matches log events based on whether the specified log field is present in the log event. The

`isPresent` works only with values in the leaf node of JSON properties. For example, a filter that looks for matches to `c-count` does not evaluate a log event with a value of `details.c-count.c1`.

See the following four filter examples:

```
{"Match": "$.httpMethod", "In": [ "PUT", ] }
{"Match": "$.StatusCode", "EqualTo": 200 }
{"Match": "$.BytesReceived", "GreaterThan": 10000}
{"Match": "$.eventSource", "StartsWith": [ "ec2", "ecs" ] }
```

AggregateOn

Valid values are `Count` and `Sum`. Specifies whether to aggregate the report based on a count of occurrences or a sum of the values of the field that is specified in the `ValueOf` field.

JSON property format notation

The `Keys`, `ValueOf`, and `Match` fields follow JSON property format with dot notation, where $ represents the root of the JSON object. This is followed by a period and then an alphanumeric string with the name of the subproperty. Multiple property levels are supported.

The first character of the string can only be A-Z or a-z. The following characters of the string can be A-Z, a-z, or 0-9.

The following list illustrates valid examples of JSON property format:

```
$.userAgent
$.endpoints[0]
$.users[1].name
$.requestParameters.instanceId
```

Additional field in rules for CLF logs

Common Log Format (CLF) log events do not have names for the fields like JSON does. To provide the fields to use for Contributor Insights rules, a CLF log event can be treated as array with an index starting from 1. You can specify the first field as **"1"**, the second field as **"2"**, and so on.

To make a rule for a CLF log easier to read, you can use `Fields`. This enables you to provide a naming alias for CLF field locations. For example, you can specify that the location "4" is an IP

address. Once specified, `IpAddress` can be used as property in the `Keys`, `ValueOf`, and `Filters` in the rule.

The following is an example of a rule for a CLF log that uses the `Fields` field.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "API-Gateway-Access-Logs*"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "IpAddress",
        "7": "StatusCode"
    },
    "Contribution": {
        "Keys": [
            "IpAddress"
        ],
        "Filters": [
            {
                "Match": "StatusCode",
                "EqualTo": 200
            }
        ]
    },
    "AggregateOn": "Count"
}
```

CloudWatch Contributor Insights rule examples

This section contains examples that illustrate use cases for Contributor Insights rules.

VPC Flow Logs: Byte transfers by source and destination IP address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
```

```
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "4": "srcaddr",
        "5": "dstaddr",
        "10": "bytes"
    },
    "Contribution": {
        "Keys": [
            "srcaddr",
            "dstaddr"
        ],
        "ValueOf": "bytes",
        "Filters": []
    },
    "AggregateOn": "Sum"
}
```

VPC Flow Logs: Highest number of HTTPS requests

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "5": "destination address",
        "7": "destination port",
        "9": "packet count"
    },
    "Contribution": {
        "Keys": [
            "destination address"
        ],
        "ValueOf": "packet count",
        "Filters": [
            {
```

```
                "Match": "destination port",
                "EqualTo": 443
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

VPC Flow Logs: Rejected TCP connections

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
```

```
    "AggregateOn": "Sum"
}
```

Route 53 NXDomain responses by source address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.rcode",
                "StartsWith": [
                    "NXDOMAIN"
                ]
            }
        ],
        "Keys": [
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 resolver queries by domain name

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_name"
        ]
```

```
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Route 53 resolver queries by query type and source address

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [],
        "Keys": [
            "$.query_type",
            "$.srcaddr"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "<loggroupname>"
    ]
}
```

Viewing Contributor Insights reports in CloudWatch

To view graphs of report data and a ranked list of contributors found by your rules, follow these steps.

To view your rule reports

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Logs**, **Contributor Insights**.
3. In the list of rules, choose the name of a rule.

   The graph displays the results of the rule over the last three hours. The table under the graph shows the top 10 contributors.

4. To change the number of contributors shown in the table, choose **Top 10 contributors** at the top of the graph.
5. To filter the graph to show only the results from a single contributor, choose that contributor in the table legend. To again show all contributors, choose that same contributor again in the legend.
6. To change the time range shown in the report, choose **15m**, **30m**, **1h**, **2h**, **3h**, or **custom** at the top of the graph.

   The maximum time range for the report is 24 hours, but you can choose a 24-hour window that occurred up to 15 days ago. To choose a time window in the past, choose **custom**, **absolute**, and then specify your time window.
7. To change the length of the time period used for the aggregation and ranking of contributors, choose **period** at the top of the graph. Viewing a longer time period generally shows a smoother report with few spikes. Choosing a shorter time period is more likely to display spikes.
8. To add this graph to a CloudWatch dashboard, choose **Add to dashboard**.
9. To open the CloudWatch Logs Insights query window, with the log groups in this report already loaded in the query box, choose **View logs**.
10. To export the report data to your clipboard or a CSV file, choose **Export**.

Graphing metrics generated by rules in CloudWatch

Contributor Insights provides a metric math function, `INSIGHT_RULE_METRIC`. You can use this function to add data from a Contributor Insights report to a graph in the **Metrics** tab of the CloudWatch console. You can also set an alarm based on this math function. For more information about metric math functions, see Math expressions with metrics.

To use this metric math function, you must be signed in to an account that has both the `cloudwatch:GetMetricData` and `cloudwatch:GetInsightRuleReport` permissions.

The syntax is `INSIGHT_RULE_METRIC(`*`ruleName`*`, `*`metricName`*`)`. *`ruleName`* is the name of a Contributor Insights rule. *`metricName`* is one of the values in the following list. The value of *`metricName`* determines which type of data the math function returns.

- `UniqueContributors` — the number of unique contributors for each data point.

- `MaxContributorValue` — the value of the top contributor for each data point. The identity of the contributor might change for each data point in the graph.

  If this rule aggregates by `Count`, the top contributor for each data point is the contributor with the most occurrences in that period. If the rule aggregates by `Sum`, the top contributor is the contributor with the greatest sum in the log field specified by the rule's `Value` during that period.
- `SampleCount` — the number of data points matched by the rule.
- Sum — the sum of the values from all contributors during the time period represented by that data point.
- `Minimum` — the minimum value from a single observation during the time period represented by that data point.
- `Maximum` — the maximum value from a single observation during the time period represented by that data point.
- `Average` — the average value from all contributors during the time period represented by that data point.

Setting an alarm on Contributor Insights metric data

Using the function `INSIGHT_RULE_METRIC`, you can set alarms on metrics that Contributor Insights generates. For example, you can create an alarm that's based on the percentage of rejected transmission control protocol (TCP) connections. To get started with this type of alarm, you can create rules like the ones shown in the following two examples:

Example rule: "RejectedConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
```

```
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [
            {
                "Match": "protocol",
                "EqualTo": 6
            },
            {
                "Match": "action",
                "In": [
                    "REJECT"
                ]
            }
        ]
    },
    "AggregateOn": "Sum"
}
```

Example rule: "TotalConnectionsRule"

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "LogGroupNames": [
        "/aws/containerinsights/sample-cluster-name/flowlogs"
    ],
    "LogFormat": "CLF",
    "Fields": {
        "3": "interfaceID",
        "4": "sourceAddress",
        "8": "protocol",
        "13": "action"
    },
    "Contribution": {
        "Keys": [
```

```
            "interfaceID",
            "sourceAddress"
        ],
        "Filters": [{
            "Match": "protocol",
            "EqualTo": 6
        }],
        "AggregateOn": "Sum"
    }
}
```

After you create your rules, you can select the **Metrics** tab in the CloudWatch Console, where you can use the following example metric math expressions to graph the data that Contributor Insights reports:

Example: Metric math expressions

```
e1 INSIGHT_RULE_METRIC("RejectedConnectionsRule", "Sum")
e2 INSIGHT_RULE_METRIC("TotalConnectionsRule", "Sum")
e3 (e1/e2)*100
```

In the example, the metric math expression `e3` returns all of the rejected TCP connections. If you want to be notified when 20 percent of the TCP connections are rejected, you can modify the expression by changing the threshold from `100` to `20`.

> **Note**
>
> You can set an alarm on a metric that you're monitoring from the **Metrics** section. While on the **Graphed metrics** tab, you can select the **Create alarm** icon under the **Actions** column. The **Create alarm** icon looks like a bell.

For more information about graphing metrics and using metric math functions, see the following section: Add a math expression to a CloudWatch graph.

Using Contributor Insights built-in rules in CloudWatch

You can use Contributor Insights built-in rules to analyze metrics from other AWS services. The following services support built-in rules:

- Contributor Insights for Amazon DynamoDB in the *Amazon DynamoDB Developer Guide*.
- Use built-in Contributor Insights rules in the *AWS PrivateLink Guide*.

Detect common application problems with CloudWatch Application Insights

You can use Amazon CloudWatch Application Insights to detect problems with your applications. CloudWatch Application Insights facilitates observability for your applications and underlying AWS resources. It helps you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications. Application Insights, which is powered by SageMaker and other AWS technologies, provides automated dashboards that show potential problems with monitored applications, which help you to quickly isolate ongoing issues with your applications and infrastructure. The enhanced visibility into the health of your applications that Application Insights provides helps reduce mean time to repair (MTTR) to troubleshoot your application issues.

When you add your applications to Amazon CloudWatch Application Insights, it scans the resources in the applications and recommends and configures metrics and logs on CloudWatch for application components. Example application components include SQL Server backend databases and Microsoft IIS/Web tiers. Application Insights analyzes metric patterns using historical data to detect anomalies, and continuously detects errors and exceptions from your application, operating system, and infrastructure logs. It correlates these observations using a combination of classification algorithms and built-in rules. Then, it automatically creates dashboards that show the relevant observations and problem severity information to help you prioritize your actions. For common problems in .NET and SQL application stacks, such as application latency, SQL Server failed backups, memory leaks, large HTTP requests, and canceled I/O operations, it provides additional insights that point to a possible root cause and steps for resolution. Built-in integration with AWS SSM OpsCenter allows you to resolve issues by running the relevant Systems Manager Automation document.

Sections

- What is Amazon CloudWatch Application Insights?
- How Amazon CloudWatch Application Insights works
- Prerequisites, IAM policies, and permissions needed to access CloudWatch Application Insights
- Set up application for monitoring using the AWS Management Console

- Application Insights cross-account observability
- Work with component configurations
- Create and configure CloudWatch Application Insights monitoring using CloudFormation templates
- Tutorial: Set up monitoring for SAP ASE
- Tutorial: Set up monitoring for SAP HANA
- Tutorial: Set up monitoring for SAP NetWeaver
- View and troubleshoot problems detected by Amazon CloudWatch Application Insights
- Logs and metrics supported by Amazon CloudWatch Application Insights

What is Amazon CloudWatch Application Insights?

CloudWatch Application Insights helps you monitor your applications that use Amazon EC2 instances along with other application resources. It identifies and sets up key metrics, logs, and alarms across your application resources and technology stack (for example, your Microsoft SQL Server database, web (IIS) and application servers, OS, load balancers, and queues). It continuously monitors metrics and logs to detect and correlate anomalies and errors. When errors and anomalies are detected, Application Insights generates CloudWatch Events that you can use to set up notifications or take actions. To assist with troubleshooting, it creates automated dashboards for detected problems, which include correlated metric anomalies and log errors, along with additional insights to point you to a potential root cause. The automated dashboards help you to take remedial actions to keep your applications healthy and to prevent impact to the end-users of your application. It also creates OpsItems so that you can resolve problems using AWS SSM OpsCenter.

You can configure important counters, such as Mirrored Write Transaction/sec, Recovery Queue Length, and Transaction Delay, as well as Windows Event Logs on CloudWatch. When a failover event or problem occurs with your SQL HA workload, such as a restricted access to query a target database, CloudWatch Application Insights provides automated insights .

CloudWatch Application Insights integrates with AWS Launch Wizard to provide a one-click monitoring setup experience for deploying SQL Server HA workloads on AWS. When you select the option to set up monitoring and insights with Application Insights on the Launch Wizard console, CloudWatch Application Insights automatically sets up relevant metrics, logs, and alarms on CloudWatch, and starts monitoring newly deployed workloads. You can view automated

insights and detected problems, along with the health of your SQL Server HA workloads, on the CloudWatch console.

Contents

- Features
- Concepts
- Pricing
- Related services
- Supported application components
- Supported technology stacks

Features

Application Insights provides the following features.

Automatic set up of monitors for application resources

CloudWatch Application Insights reduces the time it takes to set up monitoring for your applications. It does this by scanning your application resources, providing a customizable list of recommended metrics and logs, and setting them up on CloudWatch to provide necessary visibility into your application resources, such as Amazon EC2 and Elastic Load Balancers (ELB). It also sets up dynamic alarms on monitored metrics. The alarms are automatically updated based on anomalies detected in the previous two weeks.

Problem detection and notification

CloudWatch Application Insights detects signs of potential problems with your application, such as metric anomalies and log errors. It correlates these observations to surface potential problems with your application. It then generates CloudWatch Events, which can be configured to receive notifications or take actions. This eliminates the need for you to create individual alarms on metrics or log errors. Additionally, you can Configure Amazon SNS notifications to receive alerts for detected problems.

Troubleshooting

CloudWatch Application Insights creates CloudWatch automatic dashboards for problems that are detected. The dashboards show details about the problem, including the associated metric

anomalies and log errors to help you with troubleshooting. They also provide additional insights that point to potential root causes of the anomalies and errors.

Concepts

The following concepts are important for understanding how Application Insights monitors your application.

Component

An auto-grouped, standalone, or custom grouping of similar resources that make up an application. We recommend grouping similar resources into custom components for better monitoring.

Observation

An individual event (metric anomaly, log error, or exception) that is detected with an application or application resource.

Problem

Problems are detected by correlating, classifying, and grouping related observations.

For definitions of other key concepts for CloudWatch Application Insights, see Amazon CloudWatch Concepts.

Pricing

CloudWatch Application Insights sets up recommended metrics and logs for selected application resources using CloudWatch metrics, Logs, and Events for notifications on detected problems. These features are charged to your AWS account according to CloudWatch pricing. For detected problems, SSM OpsItems are also created by Application Insights to notify you about problems. Additionally, Application Insights creates SSM Parameter Store parameters to configure the CloudWatch agents on your instances. The Amazon EC2 Systems Manager features are charged according to SSM pricing. You are not charged for setup assistance, monitoring, data analysis, or problem detection.

Costs for CloudWatch Application Insights

Costs for Amazon EC2 include usage of the following features:

- CloudWatch Agent
  - CloudWatch Agent log groups

- CloudWatch Agent metrics
- Prometheus log groups (for JMX workloads)

Costs for all resources include usage of the following features:

- CloudWatch alarms (majority of cost)
- SSM OpsItems (minimal cost)

Example cost calculation

The costs in this example are considered according to the following scenario.

You created a resource group that includes the following:

- An Amazon EC2 instance with SQL Server installed.
- An attached Amazon EBS volume.

When you onboard this resource group with CloudWatch Application Insights, the SQL Server workload installed on the Amazon EC2 instance is detected. CloudWatch Application Insights starts monitoring the following metrics.

The following metrics are monitored for the SQL Server instance:

- CPUUtilization
- StatusCheckFailed
- Memory % Committed Bytes in Use
- Memory Available Mbytes
- Network Interface Bytes Total/sec
- Paging File % Usage
- Physical Disk % Disk Time
- Processor % Processor Time
- SQLServer:Buffer Manager cache hit ratio
- SQLServer:Buffer Manager life expectancy
- SQLServer:General Statistics Processes blocked
- SQLServer:General Statistics User Connections

- SQLServer:Locks Number of Deadlocks/sec
- SQLServer:SQL Statistics Batch Requests/sec
- System Processor Queue Length

The following metrics are monitored for the volumes attached to the SQL Server instance:

- VolumeReadBytes
- VolumeWriteBytes
- VolumeReadOps
- VolumeWriteOps
- VolumeTotalReadTime
- VolumeTotalWriteTime
- VolumeIdleTime
- VolumeQueueLength
- VolumeThroughputPercentage
- VolumeConsumedReadWriteOps
- BurstBalance

For this scenario, the costs are calculated according to the CloudWatch pricing page and the SSM pricing page:

- **Custom metrics**

  For this scenario, 13 of the above metrics are emitted to CloudWatch using the CloudWatch agent. These metrics are treated as custom metrics. The cost for each custom metric is $.3/month. The total cost for these custom metrics is 13 * $.3 = $3.90/month.

- **Alarms**

  For this scenario, CloudWatch Application Insights monitors 26 metrics in total, which creates 26 alarms. The cost for each alarm is $.1/month. The total cost for alarms is 26 * $.1 = $2.60/month.

- **Data ingestion and error logs**

  The cost of data ingestion is $.05/GB and storage for the SQL Server error log is $.03/GB. The total cost for data ingestion and the error log is $.05/GB + $.03/GB= $.08/GB.

- **Amazon EC2 Systems Manager OpsItems**

  An SSM OpsItem is created for each problem detected by CloudWatch Application Insights. For *n* number of problems in your application, the total cost is $.00267 * *n*/month.

Related services

The following services are used along with CloudWatch Application Insights:

Related AWS services

- **Amazon CloudWatch** provides system-wide visibility into resource utilization, application performance, and operational health. It collects and tracks metrics, sends alarm notifications, automatically updates resources that you are monitoring based on the rules that you define, and allows you to monitor your own custom metrics. CloudWatch Application Insights is initiated through CloudWatch—specifically, within the CloudWatch default operational dashboards. For more information, see the *Amazon CloudWatch User Guide*.
- **CloudWatch Container Insights** collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices. You can use Container Insights to monitor Amazon ECS, Amazon Elastic Kubernetes Service, and Kubernetes platforms on Amazon EC2. When Application Insights is enabled on the Container Insights or Application Insights consoles, Application Insights displays detected problems on your Container Insights dashboard. For more information, see Container Insights .
- **Amazon DynamoDB** is a fully managed NoSQL database service that lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.
- **Amazon EC2** provides scalable computing capacity in the AWS Cloud. You can use Amazon EC2 to launch as many or as few virtual servers as you need, to configure security and networking, and to manage storage. You can scale up or down to handle changes in requirements or spikes in popularity, which reduces your need to forecast traffic. For more information, see the *Amazon EC2 User Guide for Linux Instances* or *Amazon EC2 Guide for Windows Instances*.
- **Amazon Elastic Block Store (Amazon EBS)** provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes behave like raw, unformatted block devices. You can mount these volumes as devices on your instances. Amazon EBS volumes that are attached to an instance are exposed as storage volumes that persist independently from the life of the

instance. You can create a file system on top of these volumes, or use them in any way you would use a block device (such as a hard drive). You can dynamically change the configuration of a volume attached to an instance. For more information, see the Amazon EBS User Guide.

- **Amazon EC2 Auto Scaling** helps make sure that you have the correct number of EC2 instances available to handle the load for your application. For more information, see the *Amazon EC2 Auto Scaling User Guide*.
- **Elastic Load Balancing** distributes incoming applications or network traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in multiple Availability Zones. For more information, see the *Elastic Load Balancing User Guide*.
- **IAM** is a web service that helps you to securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication), and to control the resources they can use and how they can use them (authorization). For more information, see Authentication and Access Control for Amazon CloudWatch.
- **AWS Lambda** lets you build serverless applications composed of functions that are triggered by events and automatically deploy them using CodePipeline and AWS CodeBuild. For more information, see AWS Lambda Applications.
- **AWS Launch Wizard for SQL Server** reduces the time it takes to deploy SQL Server high availability solution to the cloud. You input your application requirements, including performance, number of nodes, and connectivity on the service console, and AWS Launch Wizard identifies the right AWS resources to deploy and run your SQL Server Always On application.
- **AWS Resource Groups** help you to organize the resources that make up your application. With Resource Groups, you can manage and automate tasks on a large number of resources at one time. Only one Resource Group can be registered for a single application. For more information, see the *AWS Resource Groups User Guide*.
- **Amazon SQS** offers a secure, durable, and available hosted queue that allows you to integrate and decouple distributed software systems and components. For more information, see the *Amazon SQS User Guide*.
- **AWS Step Functions** is a serverless function composer that allows you to sequence a variety of AWS services and resources, including AWS Lambda functions, into structured, visual workflows. For more information, see the AWS Step Functions User Guide.
- **AWS SSM OpsCenter** aggregates and standardizes OpsItems across services while providing contextual investigation data about each OpsItem, related OpsItems, and related resources. OpsCenter also provides Systems Manager Automation documents (runbooks) that you can use to quickly resolve issues. You can specify searchable, custom data for each OpsItem. You can also

view automatically-generated summary reports about OpsItems by status and source. For more information, see the *AWS Systems Manager User Guide*.

- **Amazon API Gateway** is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. For more information, see the Amazon API Gateway User Guide.

  > **Note**
  >
  > Application Insights supports only REST API protocols (v1 of the API Gateway service).

- **Amazon Elastic Container Service (Amazon ECS)** is a fully managed container orchestration service. You can use Amazon ECS to run your most sensitive and mission-critical applications. For more information, see the Amazon Elastic Container Service Developer Guide.
- **Amazon Elastic Kubernetes Service (Amazon EKS)** is a managed service that you can use to run Kubernetes on AWS without having to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. For more information, see the Amazon EKS User Guide.
- **Kubernetes on Amazon EC2**. Kubernetes is open-source software that helps you deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. With Kubernetes, you can run any type of containerized application with the same toolset on-premises and in the cloud. For more information, see Kubernetes Documentation: Getting started.
- **Amazon FSx** helps you to launch and run popular file systems that are fully managed by AWS. With Amazon FSx, you can leverage the feature sets and performance of common open source and commercially-licensed file systems to avoid time-consuming administrative tasks. For more information, see the Amazon FSx Documentation.
- **Amazon Simple Notification Service (SNS)** is a fully-managed messaging service for both application-to-application and application-to-person communication. You can configure Amazon SNS for monitoring by Application Insights. When Amazon SNS is configured as a resource for monitoring, Application Insights tracks SNS metrics to help determine why SNS messages may encounter issues or fail.
- **Amazon Elastic File System (Amazon EFS)** is a fully-managed elastic NFS file system for use with AWS Cloud services and on-premises resources. It is built to scale to petabytes on demand

without disrupting applications. It grows and shrinks automatically as you add and remove files, which eliminates the need to provision and manage capacity to accommodate growth. For more information, see the Amazon Elastic File System documentation.

Related third-party services

- For some workloads and applications monitored in Application Insights, **Prometheus JMX exporter** is installed using AWS Systems Manager Distributor so that CloudWatch Application Insights can retrieve Java-specific metrics. When you choose to monitor a Java application, Application Insights automatically installs the Prometheus JMX exporter for you.

Supported application components

CloudWatch Application Insights scans your resource group to identify application components. Components can be standalone, auto-grouped (such as instances in an Auto Scaling group or behind a load balancer), or custom (by grouping together individual Amazon EC2 instances).

The following components are supported by CloudWatch Application Insights:

AWS components

- Amazon EC2
- Amazon EBS
- Amazon RDS
- Elastic Load Balancing: Application Load Balancer and Classic Load Balancer (all target instances of these load balancers are identified and configured).
- Amazon EC2 Auto Scaling groups: AWS Auto Scaling (Auto Scaling groups are dynamically configured for all target instances; if your application scales up, CloudWatch Application Insights automatically configures the new instances). Auto Scaling groups are not supported for CloudFormation stack-based resource groups.
- AWS Lambda
- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB table
- Amazon S3 bucket metrics
- AWS Step Functions

- Amazon API Gateway REST API stages
- Amazon Elastic Container Service (Amazon ECS): cluster, service, and task
- Amazon Elastic Kubernetes Service (Amazon EKS): cluster
- Kubernetes on Amazon EC2: Kubernetes cluster running on EC2
- Amazon SNS topic

Any other component type resources are not currently tracked by CloudWatch Application Insights. If a component type that is supported does not appear in your Application Insights application, the component may already be registered and managed by another application you own that is monitored by Application Insights.

Supported technology stacks

You can use CloudWatch Application Insights to monitor your applications running on Windows Server and Linux operating systems by selecting the application tier dropdown menu option for one of the following technologies:

- Front-end: Microsoft Internet Information Services (IIS) Web Server
- Worker-tier:
  - .NET Framework
  - .NET Core
- Applications:
  - Java
  - SAP NetWeaver standard, distributed, and high availability deployments
- Active Directory
- SharePoint
- Databases:
  - Microsoft SQL Server running on Amazon RDS or Amazon EC2 (including SQL Server High Availability configurations. See, Component configuration examples).
  - MySQL running on Amazon RDS, Amazon Aurora, or Amazon EC2
  - PostgreSQL running on Amazon RDS or Amazon EC2
  - Amazon DynamoDB table
  - Oracle running on Amazon RDS or Amazon EC2

- SAP HANA database on a single Amazon EC2 instance and multiple EC2 instances
- Cross-AZ SAP HANA database high availability setup
- SAP Sybase ASE database on a single Amazon EC2 instance
- Cross-AZ SAP Sybase ASE database high availability setup

If none of the technology stacks listed above apply to your application resources, you can monitor your application stack by choosing **Custom** from the application tier dropdown menu on the **Manage monitoring** page.

How Amazon CloudWatch Application Insights works

CloudWatch Application Insights provides monitoring of your application resources. The following information describes how Application Insights works.

Topics

- How Application Insights monitors applications
- Data retention
- Quotas
- AWS Systems Manager (SSM) packages used by CloudWatch Application Insights
- AWS Systems Manager (SSM) Documents used by CloudWatch Application Insights

How Application Insights monitors applications

The following information describes how Application Insights monitors applications.

Application discovery and configuration

The first time an application is added to CloudWatch Application Insights it scans the application components to recommend key metrics, logs, and other data sources to monitor for your application. You can then configure your application based on these recommendations.

Data preprocessing

CloudWatch Application Insights continuously analyzes the data sources being monitored across the application resources to discover metric anomalies and log errors (observations).

Intelligent problem detection

The CloudWatch Application Insights engine detects problems in your application by correlating observations using classification algorithms and built-in rules. To assist in troubleshooting, it creates automated CloudWatch dashboards, which include contextual information about the problems.

Alert and action

When CloudWatch Application Insights detects a problem with your application, it generates CloudWatch Events to notify you of the problem. See Application Insights CloudWatch Events for detected problems for more information about how to set up these Events. Additionally, you can configure Amazon SNS notifications to receive alerts for detected problems.

Example scenario

You have an ASP .NET application that is backed by a SQL Server database. Suddenly, your database begins to malfunction because of high memory pressure. This leads to application performance degradation and possibly HTTP 500 errors in your web servers and load balancer.

With CloudWatch Application Insights and its intelligent analytics, you can identify the application layer that is causing the problem by checking the dynamically created dashboard that shows the related metrics and log file snippets. In this case, the problem might be at the SQL database layer.

Data retention

CloudWatch Application Insights retains problems for 55 days and observations for 60 days.

Quotas

For default quotas for CloudWatch Application Insights, see Amazon CloudWatch Application Insights endpoints and quotas. Unless otherwise noted, each quota is per AWS Region. Contact AWS Support to request an increase in your service quota. Many services contain quotas that cannot be changed. For more information about the quotas for a specific service, see the documentation for that service.

AWS Systems Manager (SSM) packages used by CloudWatch Application Insights

The packages listed in this section are used by Application Insights, and can be independently managed and deployed with AWS Systems Manager Distributor. For more information about SSM Distributor, see AWS Systems Manager Distributor in the *AWS Systems Manager User Guide*.

Packages:

- AWSObservabilityExporter-JMXExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure
- AWSObservabilityExporter-HAClusterExporterInstallAndConfigure
- AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure
- AWSObservabilityExporter-SQLExporterInstallAndConfigure

`AWSObservabilityExporter-JMXExporterInstallAndConfigure`

You can retrieve workload-specific Java metrics from Prometheus JMX exporter for Application Insights to configure and monitor alarms. In the Application Insights console, on the **Manage monitoring** page, select **JAVA application** from the **Application tier** dropdown. Then under **JAVA Prometheus exporter configuration**, select your **Collection method** and **JMX port number**.

To use AWS Systems Manager Distributor to package, install, and configure the AWS-provided Prometheus JMX exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus JMX exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- The JAVA_HOME environment variable is set

Install and configure the `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package

The `AWSObservabilityExporter-JMXExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure Prometheus JMX Exporter. When Java metrics are sent by the Prometheus JMX exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the Prometheus JMX exporter YAML configuration file located in the Prometheus GitHub repository. Use the example configuration and option descriptions to guide you.
2. Copy the Prometheus JMX exporter YAML configuration file encoded as Base64 to a new SSM parameter in SSM Parameter Store.

3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-JMXExporterInstallAndConfigure** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIGURATION": "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
  "SSM_EXPOSITION_PORT": "9404"
}
```

> **Note**
>
> Port 9404 is the default port used to send Prometheus JMX metrics. You can update this port.

Example: Configure CloudWatch agent to retrieve Java metrics

1. Install the Prometheus JMX exporter, as described in the previous procedure. Then verify that it is correctly installed on your instance by checking the port status.

   Successful installation on Windows instance example

```
PS C:\> curl http://localhost:9404 (http://localhost:9404/)
StatusCode : 200
StatusDescription : OK
Content : # HELP jvm_info JVM version info
```

   Successful installation on Linux instance example

```
$ curl localhost:9404
# HELP jmx_config_reload_failure_total Number of times configuration have failed to
 be reloaded.
# TYPE jmx_config_reload_failure_total counter
jmx_config_reload_failure_total 0.0
```

2. Create the Prometheus service discovery YAML file. The following example service discovery file performs the following:

- Specifies the Prometheus JMX exporter host port as `localhost: 9404`.
- Attaches labels (`Application`, `ComponentName`, and `InstanceId`) to the metrics, which can be set as CloudWatch metric dimensions.

```
$ cat prometheus_sd_jmx.yaml
- targets:
  - 127.0.0.1:9404
  labels:
    Application: myApp
    ComponentName: arn:aws:elasticloadbalancing:us-
east-1:123456789012:loadbalancer/app/sampl-Appli-MMZW8E3GH4H2/aac36d7fea2a6e5b
    InstanceId: i-12345678901234567
```

3. Create the Prometheus JMX exporter configuration YAML file. The following example configuration file specifies the following:

   - The metrics retrieval job interval and timeout period.
   - The metrics retrieval jobs (`jmx` and `sap`), also known as scraping, which include the job name, maximum time series returned at a time, and service discovery file path.

```
$ cat prometheus.yaml
global:
  scrape_interval: 1m
  scrape_timeout: 10s
scrape_configs:
  - job_name: jmx
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_jmx.yaml"]
  - job_name: sap
    sample_limit: 10000
    file_sd_configs:
      - files: ["/tmp/prometheus_sd_sap.yaml"]
```

4. Verify that the CloudWatch agent is installed on your Amazon EC2 instance and that the version is 1.247346.1b249759 or later. To install the CloudWatch agent on your EC2 instance, see Installing the CloudWatch Agent. To verify the version, see Finding information about CloudWatch agent versions.

5. Configure the CloudWatch agent. For more information about how to configure the CloudWatch agent configuration file, see Manually create or edit the CloudWatch agent configuration file. The following example CloudWatch agent configuration file performs the following:

   - Specifies the Prometheus JMX exporter configuration file path.
   - Specifies the target log group to which to publish EMF metric logs.
   - Specifies two sets of dimensions for each metric name.
   - Sends 8 (4 metric names * 2 sets of dimensions per metric name) CloudWatch metrics.

```
{
   "logs":{
      "logs_collected":{
         ....
      },
      "metrics_collected":{
         "prometheus":{
            "cluster_name":"prometheus-test-cluster",
            "log_group_name":"prometheus-test",
            "prometheus_config_path":"/tmp/prometheus.yaml",
            "emf_processor":{
               "metric_declaration_dedup":true,
               "metric_namespace":"CWAgent",
               "metric_unit":{
                  "jvm_threads_current":"Count",
                  "jvm_gc_collection_seconds_sum":"Second",
                  "jvm_memory_bytes_used":"Bytes"
               },
               "metric_declaration":[
                  {
                     "source_labels":[
                        "job"
                     ],
                     "label_matcher":"^jmx$",
                     "dimensions":[
                        [
                           "InstanceId",
                           "ComponentName"
                        ],
                        [
                           "ComponentName"
```

```
                        ]
                    ],
                    "metric_selectors":[
                        "^java_lang_threading_threadcount$",
                        "^java_lang_memory_heapmemoryusage_used$",
                        "^java_lang_memory_heapmemoryusage_committed$"
                    ]
                }
            ]
        }
    }
},
"metrics":{
    ....
}
}
```

`AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure`

You can retrieve workload-specific SAP HANA metrics from Prometheus HANA database exporter for Application Insights to configure and monitor alarms. For more information, see Set up your SAP HANA database for monitoring in this guide.

To use AWS Systems Manager Distributor to package, install, and configure the AWS-provided Prometheus HANA database exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HANA database exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP HANA database
- Linux operating system (SUSE Linux, RedHat Linux)
- A secret with SAP HANA database monitoring credentials, using AWS Secrets Manager. Create a secret using the key/value pairs format, specify the key username, and enter the database user for the value. Add a second key password, and then enter the password for the value. For more information about how to create secrets, see Create a secret in the *AWS Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
```

```
  "username": "<database_user>",
  "password": "<database_password>"
}
```

Install and configure the AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure package

The AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure package is an SSM Distributor package that you can use to install and configure Prometheus HANA database Exporter. When HANA database metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations. The following is an example parameter value.

```
{\"exposition_port\":9668,\"multi_tenant\":true,\"timeout\":600,\"hana\":{\"host\":
\"localhost\",\"port\":30013,\"aws_secret_name\":\"HANA_DB_CREDS\",\"scale_out_mode
\":true}}
```

> **Note**
>
> In this example, the export runs only on the Amazon EC2 instance with the active SYSTEM database, and it will remain idle on the other EC2 instances in order to avoid duplicate metrics. The exporter can retrieve all of the database tenant information from the SYSTEM database.

2. Create an SSM parameter in SSM Parameter Store to store the Exporter metrics queries. The package can accept more than one metrics parameter. Each parameter must have a valid JSON object format. The following is an example parameter value:

```
{\"SELECT MAX(TIMESTAMP) TIMESTAMP, HOST, MEASURED_ELEMENT_NAME CORE,
 SUM(MAP(CAPTION, 'User Time', TO_NUMBER(VALUE), 0)) USER_PCT, SUM(MAP(CAPTION,
 'System Time', TO_NUMBER(VALUE), 0)) SYSTEM_PCT, SUM(MAP(CAPTION, 'Wait
 Time', TO_NUMBER(VALUE), 0)) WAITIO_PCT, SUM(MAP(CAPTION, 'Idle Time', 0,
 TO_NUMBER(VALUE))) BUSY_PCT, SUM(MAP(CAPTION, 'Idle Time', TO_NUMBER(VALUE), 0))
 IDLE_PCT FROM sys.M_HOST_AGENT_METRICS WHERE MEASURED_ELEMENT_TYPE = 'Processor'
 GROUP BY HOST, MEASURED_ELEMENT_NAME;\":{\"enabled\":true,\"metrics\":[{\"name\":
\"hanadb_cpu_user\",\"description\":\"Percentage of CPU time spent by HANA DB in user
 space, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
```

```
\"USER_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_system
\",\"description\":\"Percentage of CPU time spent by HANA DB in Kernel space,
 over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":
\"SYSTEM_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_waitio
\",\"description\":\"Percentage of CPU time spent by HANA DB in IO mode, over the
 last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE\"],\"value\":\"WAITIO_PCT\",
\"unit\":\"percent\",\"type\":\"gauge\"},{\"name\":\"hanadb_cpu_busy\",\"description
\":\"Percentage of CPU time spent by HANA DB, over the last minute (in seconds)\",
\"labels\":[\"HOST\",\"CORE\"],\"value\":\"BUSY_PCT\",\"unit\":\"percent\",\"type\":
\"gauge\"},{\"name\":\"hanadb_cpu_idle\",\"description\":\"Percentage of CPU time not
 spent by HANA DB, over the last minute (in seconds)\",\"labels\":[\"HOST\",\"CORE
\"],\"value\":\"IDLE_PCT\",\"unit\":\"percent\",\"type\":\"gauge\"}]}}
```

For more information about metrics queries, see the SUSE / hanadb_exporter repo on GitHub.

3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure*** and choose **Install one time**.
4. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_EXPORTER_METRICS_1": "{{ssm:<SSM_FIRST_METRICS_PARAMETER_STORE_NAME>}}",
  "SSM_EXPORTER_METRICS_2": "{{ssm:<SSM_SECOND_METRICS_PARAMETER_STORE_NAME>}}"
}
```

5. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

`AWSObservabilityExporter-HAClusterExporterInstallAndConfigure`

You can retrieve workload-specific High Availability (HA) cluster metrics from Prometheus HANA cluster exporter for Application Insights to configure and monitor alarms for an SAP HANA database High Availability setup. For more information, see Set up your SAP HANA database for monitoring in this guide.

To use AWS Systems Manager Distributor to package, install, and configure the AWS-provided Prometheus HA cluster exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus HA cluster exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- HA cluster for Pacemaker, Corosync, SBD, and DRBD
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the `AWSObservabilityExporter-HAClusterExporterInstallAndConfigure` package

The `AWSObservabilityExporter-HAClusterExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure Prometheus HA Cluster Exporter. When cluster metrics are sent by the Prometheus HANA database exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations in JSON format. The following is an example parameter value.

   ```
   {\"port\":\"9664\",\"address\":\"0.0.0.0\",\"log-level\":\"info\",\"crm-mon-path
   \":\"/usr/sbin/crm_mon\",\"cibadmin-path\":\"/usr/sbin/cibadmin\",\"corosync-
   cfgtoolpath-path\":\"/usr/sbin/corosync-cfgtool\",\"corosync-quorumtool-path\":\"/
   usr/sbin/corosync-quorumtool\",\"sbd-path\":\"/usr/sbin/sbd\",\"sbd-config-path\":
   \"/etc/sysconfig/sbd\",\"drbdsetup-path\":\"/sbin/drbdsetup\",\"enable-timestamps
   \":false}
   ```

   For more information about the exporter configurations, see the ClusterLabs / ha_cluster_exporter repo on GitHub.

2. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-HAClusterExporterInstallAndConfigure*** and choose **Install one time**.

3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

   ```
   {
     "SSM_EXPORTER_CONFIG": "{{ssm:<*SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>*}}"
   }
   ```

4. Select the Amazon EC2 instances with SAP HANA database, and choose **Run**.

`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure`

You can retrieve workload-specific SAP NetWeaver metrics from Prometheus SAP host exporter for Application Insights to configure and monitor alarms for SAP NetWeaver Distributed and High Availability deployments.

To use AWS Systems Manager Distributor to package, install, and configure the SAP host exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SAP host exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- SAP NetWeaver application servers
- Linux operating system (SUSE Linux, RedHat Linux)

Install and configure the `AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` package

The `AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure SAP NetWeaver Prometheus metrics exporter. When SAP NetWeaver metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Create an SSM parameter in SSM Parameter Store to store the Exporter configurations in JSON format. The following is an example parameter value.

   ```
   {\"address\":\"0.0.0.0\",\"port\":\"9680\",\"log-level\":\"info\",\"is-HA\":false}
   ```

   - **address**

     The target address to which to send the Prometheus metrics. The default value is `localhost`.

   - **port**

     The target port to which to send the Prometheus metrics. The default value is `9680`.

   - **is-HA**

     `true` for SAP NetWeaver High Availability deployments. For all other deployments the value is `false`.

2. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure** and choose **Install one time**.
3. Update the SSM parameter you created in the first step by replacing "Additional Arguments" with the following:

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:<SSM_CONFIGURATIONS_PARAMETER_STORE_NAME>}}",
  "SSM_SID": "<SAP_DATABASE_SID>",
  "SSM_INSTANCES_NUM": "<instances_number seperated by comma>"
}
```

Example

```
{
  "SSM_EXPORTER_CONFIG": "{{ssm:exporter_config_paramter}}",
  "SSM_INSTANCES_NUM": "11,12,10",
  "SSM_SID": "PR1"
}
```

4. Select the Amazon EC2 instances with SAP NetWeaver applications, and choose **Run**.

> **Note**
>
> The Prometheus exporter services the SAP NetWeaver metrics on a local endpoint. The local endpoint can be accessed by only the operating system users on the Amazon EC2 instance. Therefore, after the exporter package is installed, the metrics are available to all of the operating system users. The default local endpoint is `localhost:9680/metrics`.

`AWSObservabilityExporter-SQLExporterInstallAndConfigure`

You can retrieve workload-specific SQL Server metrics from Prometheus SQL exporter for Application Insights to monitor key metrics.

To use AWS Systems Manager Distributor to package, install, and configure the SQL exporter package independently of Application Insights, complete the following steps.

Prerequisites for using the Prometheus SQL exporter SSM package

- SSM agent version 2.3.1550.0 or later installed
- Amazon EC2 instance running SQL Server on Windows with SQL Server user authentication enabled.
- A SQL Server user with the following permissions:

```
GRANT VIEW ANY DEFINITION TO
```

```
GRANT VIEW SERVER STATE TO
```

- A secret containing the database connection string using AWS Secrets Manager. For more information about how to create secrets, see Create a secret in the *AWS Secrets Manager User Guide*. The secret must be formatted as follows:

```
{
  "data_source_name":"sqlserver://<username>:<password>@localhost:1433"
}
```

> **Note**
>
> If the password or username contains special characters, you must percent encode the special characters to make sure that the connection to the database succeeds.

Install and configure the `AWSObservabilityExporter-SQLExporterInstallAndConfigure` package

The `AWSObservabilityExporter-SQLExporterInstallAndConfigure` package is an SSM Distributor package that you can use to install and configure SQL Prometheus metrics exporter. When metrics are sent by the Prometheus exporter, the CloudWatch agent can be configured to retrieve the metrics for the CloudWatch service.

1. Based on your preferences, prepare the SQL Exporter YAML configuration. The following sample configuration has a single metric configured. Use the example configuration to update the configuration with additional metrics or create your own configuration.

```
---
```

```
global:
  scrape_timeout_offset: 500ms
  min_interval: 0s
  max_connections: 3
  max_idle_connections: 3
target:
  aws_secret_name: <SECRET_NAME>
  collectors:
    - mssql_standard
collectors:
  - collector_name: mssql_standard
    metrics:
    - metric_name: mssql_batch_requests
      type: counter
      help: 'Number of command batches received.'
      values: [cntr_value]
      query: |
        SELECT cntr_value
        FROM sys.dm_os_performance_counters WITH (NOLOCK)
        WHERE counter_name = 'Batch Requests/sec'
```

2. Copy the Prometheus SQL exporter YAML configuration file encoded as Base64 to a new SSM parameter in SSM Parameter Store.
3. Navigate to the SSM Distributor console and open the **Owned by Amazon** tab. Select **AWSObservabilityExporter-SQLExporterInstallAndConfigure** and choose **Install one time**.
4. Replace the "Additional Arguments" with the following information. The SSM_PARAMETER_NAME is the name of the parameter you created in Step 2.

```
{
  "SSM_EXPORTER_CONFIGURATION":
    "{{ssm:<SSM_PARAMETER_STORE_NAME>}}",
    "SSM_PROMETHEUS_PORT": "9399",
    "SSM_WORKLOAD_NAME": "SQL"
}
```

5. Select the Amazon EC2 instance with the SQL Server database, then choose run.

AWS Systems Manager (SSM) Documents used by CloudWatch Application Insights

Application Insights uses the SSM Documents listed in this section to define the actions that AWS Systems Manager performs on your managed instances. These documents use the `Run Command` capability of Systems Manager to automate the tasks necessary for carrying out Application Insights monitoring capabilities. The run schedules for these documents are maintained by Application Insights and can't be altered.

For more information about SSM Documents, see AWS Systems Manager Documents in the *AWS Systems Manager User Guide*.

Documents managed by CloudWatch Application Insights

The following table lists the SSM documents that are managed by Application Insights.

Document name	Description	Run schedule
`AWSEC2-DetectWorkload`	Auto detects applications running in your application environment that can be set up to be monitored by Application Insights.	This document runs hourly in your application environment to get up-to-date application details.
`AWSEC2-CheckPerformanceCounterSets`	Checks whether Performance Counter namespaces are enabled on your Amazon EC2 Windows instances.	This document runs hourly in your application environment and only monitors Performance Counter metrics if the corresponding namespaces are enabled.
`AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure`	Installs and configures CloudWatch Agent based on the monitoring configuration of your application components.	This document runs every 30 minutes to make sure that the CloudWatch Agent configuration is always accurate and up-to-date. The document also runs immediately after a change is made to your application monitoring setup

Document name	Description	Run schedule
		such as adding or removing metrics or updating log configurations.

Documents managed by AWS Systems Manager

The following documents are used by CloudWatch Application Insights and managed by Systems Manager.

`AWS-ConfigureAWSPackage`

Application Insights uses this document to install and uninstall Prometheus exporter distributor packages, to collect workload specific metrics, and to enable comprehensive monitoring of workloads on customer Amazon EC2 instances. CloudWatch Application Insights installs the Prometheus exporter distributor packages only if the correlated target workload is running on your instance.

The following table lists the Prometheus exporter distributor packages and the correlated target workloads.

Prometheus exporter distributor package name	Target workload
`AWSObservabilityExporter-HAClusterExporterInstallAndConfigure`	SAP HANA HA
`AWSObservabilityExporter-JMXExporterInstallAndConfigure`	Java/JMX
`AWSObservabilityExporter-SAP-HANADBExporterInstallAndConfigure`	SAP HANA
`AWSObservabilityExporter-SAP-SAPHostExporterInstallAndConfigure`	NetWeaver

Prometheus exporter distributor package name	Target workload
`AWSObservabilityExporter-SQLExporterInstallAndConfigure`	SQL Server (Windows) and SAP ASE (Linux)

`AmazonCloudWatch-ManageAgent`

Application Insights uses this document to manage the status and configuration of CloudWatch Agent on your instances and to collect internal system level metrics and logs from Amazon EC2 instances across operating systems.

Prerequisites, IAM policies, and permissions needed to access CloudWatch Application Insights

To get started with CloudWatch Application Insights, verify that you have met the following prerequisites, have created an IAM policy, and have attached permissions if needed.

Topics

- Prerequisites to configure an application for monitoring
- IAM policy for CloudWatch Application Insights
- IAM role permissions for account-based application onboarding

Prerequisites to configure an application for monitoring

You must complete the following prerequisites to configure an application with CloudWatch Application Insights:

- **AWS Systems Manager enablement** – Install Systems Manager Agent (SSM Agent) on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see Setting up AWS Systems Manager in the *AWS Systems Manager User Guide*.
- **EC2 instance role** – You must attach the following Amazon EC2 instance roles to enable Systems Manager
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see AWS Systems Manager identity-based policy examples.

  - You must attach the `CloudWatchAgentServerPolicy` policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with CloudWatch agent.
- **AWS resource groups** – To onboard your applications to CloudWatch Application Insights, create a resource group that includes all of the associated AWS resources used by your application stack. This includes application load balancers, Amazon EC2 instances running IIS and web front-end, .NET worker tiers, and SQL Server databases. For more information about application components and technology stacks supported by Application Insights, see Supported application components. CloudWatch Application Insights automatically includes Auto Scaling groups using the same tags or CloudFormation stacks as your resource group, because Auto Scaling groups are not supported by CloudFormation resource groups. For more information, see Getting Started with AWS Resource Groups.
- **IAM permissions** – For users who don't have administrative access, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see IAM policy for CloudWatch Application Insights.
- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.
- **Performance Counter metrics support for EC2 Windows instances** – To monitor Performance Counter metrics on your Amazon EC2 Windows instances, Performance Counters must be installed on the instances. For Performance Counter metrics and corresponding Performance Counter set names, see Performance Counter metrics. For more information about Performance Counters, see Performance Counters.
- **Amazon CloudWatch agent** – Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application Insights from the existing CloudWatch agent configuration file. For more information, see Manually create or edit the CloudWatch agent configuration file.

IAM policy for CloudWatch Application Insights

To use CloudWatch Application Insights, you must create an AWS Identity and Access Management (IAM) policy and attach it to your user, group, or role. For more information about users, groups,

and roles, see IAM Identities (users, user groups, and roles). The IAM policy defines the user permissions.

To create an IAM policy using the console

To create an IAM policy using the IAM console, perform the following steps.

1. Open the IAM console. In the left navigation pane, select **Policies**.
2. At the top of the page, select **Create policy**.
3. Select the **JSON** tab.
4. Copy and paste the following JSON document under the **JSON** tab.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "applicationinsights:*",
                "iam:CreateServiceLinkedRole",
                "iam:ListRoles",
                "resource-groups:ListGroups"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

5. Select **Review Policy**.
6. Enter a **Name** for the policy, for example, “AppInsightsPolicy.” Optionally, enter a **Description**.
7. Select **Create Policy**.
8. In the left navigation pane, select **User groups**, **Users**, or **Roles**.
9. Select the name of the user group, user, or role to which you would like to attach the policy.
10. Select **Add permissions**.
11. Select **Attach existing policies directly**.

12. Search for the policy that you just created, and select the check box to the left of the policy name.
13. Select **Next: Review**.
14. Make sure that the correct policy is listed, and select **Add permissions**.
15. Make sure that you log in with the user associated with the policy that you just created when you use CloudWatch Application Insights.

To create an IAM policy using the AWS CLI

To create an IAM policy using the AWS CLI, run the create-policy operation from the command line using the JSON document above as a file in your current folder.

To create an IAM policy using AWS Tools for Windows PowerShell

To create an IAM policy using the AWS Tools for Windows PowerShell, run the New-IAMPolicy cmdlt using the JSON document above as a file in your current folder.

IAM role permissions for account-based application onboarding

If you want to onboard all of the resources in your account, and you choose not to use the Application Insights managed policy for full access to Application Insights functionality, you must attach the following permissions to your IAM role so that Application Insights can discover all of the resources in your account:

```
"ec2:DescribeInstances"
"ec2:DescribeNatGateways"
"ec2:DescribeVolumes"
"ec2:DescribeVPCs"
"rds:DescribeDBInstances"
"rds:DescribeDBClusters"
"sqs:ListQueues"
"elasticloadbalancing:DescribeLoadBalancers"
"autoscaling:DescribeAutoScalingGroups"
"lambda:ListFunctions"
"dynamodb:ListTables"
"s3:ListAllMyBuckets"
"sns:ListTopics"
"states:ListStateMachines"
"apigateway:GET"
"ecs:ListClusters"
"ecs:DescribeTaskDefinition"
```

```
"ecs:ListServices"
"ecs:ListTasks"
"eks:ListClusters"
"eks:ListNodegroups"
"fsx:DescribeFileSystems"
"route53:ListHealthChecks"
"route53:ListHostedZones"
"route53:ListQueryLoggingConfigs"
"route53resolver:ListFirewallRuleGroups"
"route53resolver:ListFirewallRuleGroupAssociations"
"route53resolver:ListResolverEndpoints"
"route53resolver:ListResolverQueryLogConfigs"
"route53resolver:ListResolverQueryLogConfigAssociations"
"logs:DescribeLogGroups"
"resource-explorer:ListResources"
```

Set up application for monitoring using the AWS Management Console

This section provides steps to set up, configure, and manage your CloudWatch Application Insights application using the console, the AWS CLI, and AWS Tools for Windows PowerShell.

Topics

- Set up, configure, and manage your application for monitoring from the CloudWatch console
- Set up, configure, and manage your application for monitoring using the command line
- Application Insights CloudWatch Events for detected problems
- Receive notifications for detected problems

Set up, configure, and manage your application for monitoring from the CloudWatch console

This section provides steps to set up, configure, and manage your application for monitoring from the CloudWatch console.

Console procedures

- Add and configure an application
- Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring
- Disable monitoring for an application component
- Delete an application

Add and configure an application

Add and configure an application from the CloudWatch console

To get started with CloudWatch Application Insights from the CloudWatch console, perform the following steps.

1. **Start.** Open the CloudWatch console landing page. From the left navigation pane, under **Insights**, choose **Application Insights**. The page that opens shows the list of applications that are monitored with CloudWatch Application Insights, along with their monitoring status.
2. **Add an application.** To set up monitoring for your application, choose **Add an application**. When you choose **Add an application**, you are prompted to **Choose Application Type**.

   - **Resource group-based application**. When you select this option, you can choose which resource groups in this account to monitor. To use multiple applications on a component, you must use resource group-based monitoring.
   - **Account-based application**. When you select this option, you can monitor all of the resources in this account. If you want to monitor all of the resources in an account, we recommend this option over the resource group-based option because the application onboarding process is faster.

   > **Note**
   >
   > You can't combine resource group-based monitoring with account-based monitoring using Application Insights. In order to change the application type, you must delete all of the applications that are being monitored, and **Choose Application Type**.

   When you add your first application for monitoring, CloudWatch Application Insights creates a service-linked role in your account, which gives Application Insights permissions to call other AWS services on your behalf. For more information about the service-linked role created in your account by Application Insights, see Using service-linked roles for CloudWatch Application Insights.
3. Resource-based application monitoring

   1. **Select an application or resource group.** On the **Specify application details** page, select the AWS resource group that contains your application resources from the

dropdown list. These resources include front-end servers, load balancers, auto scaling groups, and database servers.

If you have not created a resource group for your application, you can create one by choosing **Create new resource group**. For more information about creating resource groups, see the *AWS Resource Groups User Guide*.

2. **Notifications for problem insights.** To view and get notified when problems are detected for selected applications, choose Amazon SNS notifications or Systems Manager OpsCenter Opsitems.
   a. **Set up Amazon SNS notification (Recommended).** Choose **Select existing topic** or **Create new topic**.
   b. **Integrate with AWS Systems Manager OpsCenter.** Under **Advanced Settings**, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the Amazon SNS topic ARN.
3. **Monitor CloudWatch Events**. Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
4. **Tags — optional**. CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see Working with Tag Editor.
5. Choose **Next**.

   An ARN is generated for the application in the following format.

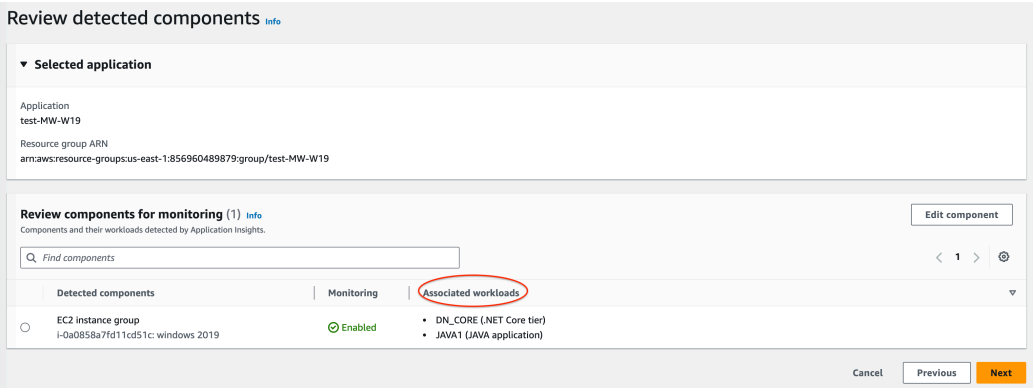
   ```
   arn:partition:applicationinsights:region:account-id:application/resource-group/resource-group-name
   ```

   Example

   ```
   arn:aws:applicationinsights:us-east-1:123456789012:application/resource-group/my-resource-group
   ```

6. On the **Review detected components** page, under **Review components for monitoring**, the table lists the detected components and their associated detected workloads.

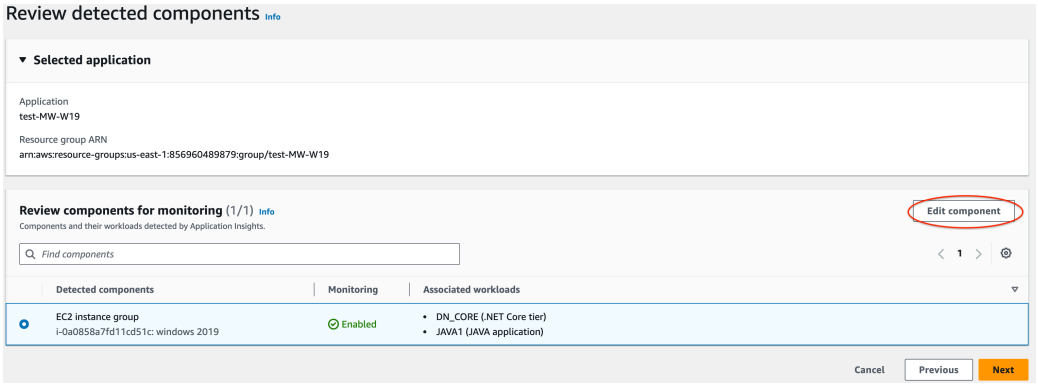
> **Note**
>
> For components that support multiple customized workloads, you can monitor up to five workloads for each component. These workloads will be monitored separately from the component.



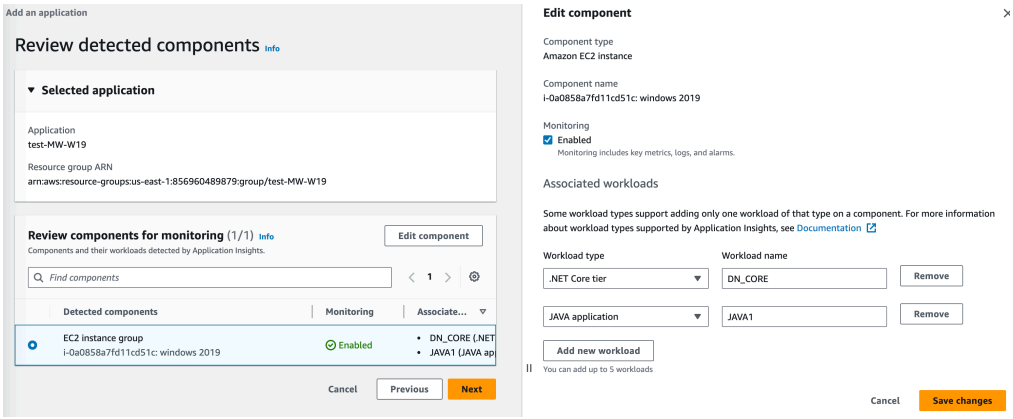
Under **Associated workloads**, there are several possible messages that appear if a workload is not listed.

- **Couldn't detect workloads** – An issue occurred when trying to detect workloads. Make sure that you have completed the Prerequisites to configure an application for monitoring. If you need to add workloads, choose **Edit component**.
- **No workloads detected** – We didn't detect any workloads. You may need to add workloads. To do so, choose **Edit component**.
- **Not applicable** – The component doesn't support customized workloads and will be monitored with default metrics, alarms, and logs. You can't add workloads to these components.

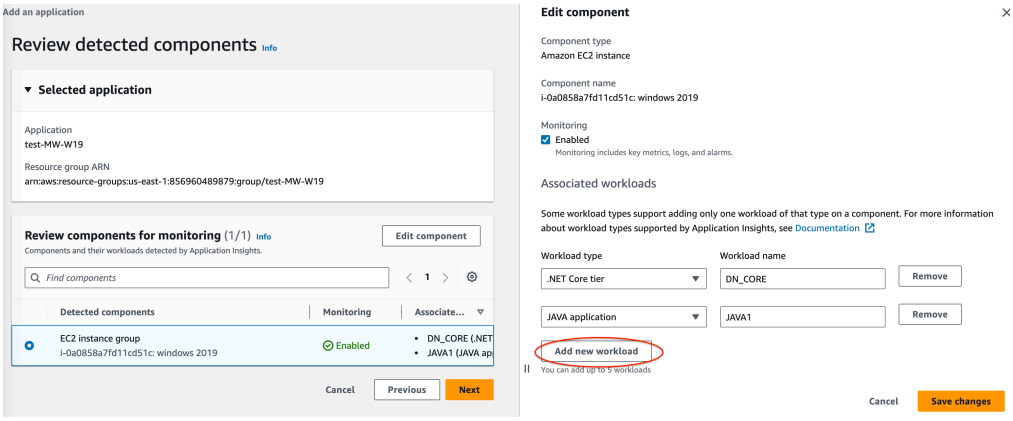
7. To edit a component, select a component, and then choose **Edit component**. A side panel opens with workloads detected on the component. In this panel, you can edit the component details and add new workloads.



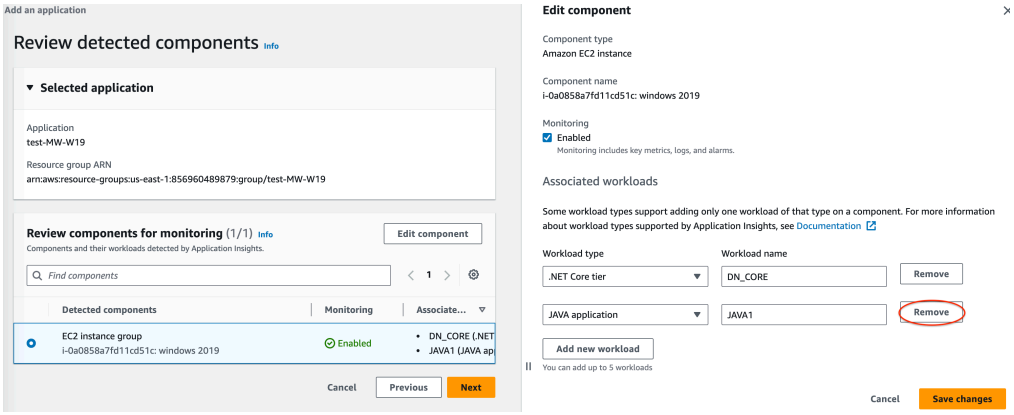
- To edit the workload type or name, use the dropdown list.



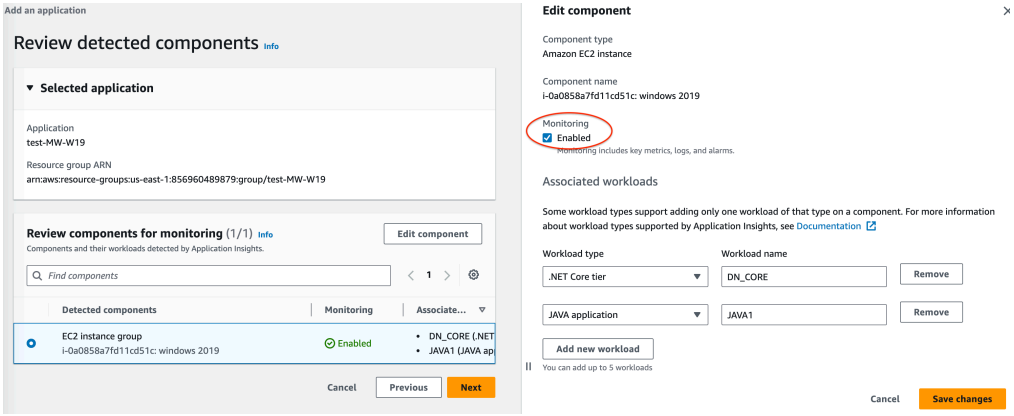
- To add a workload to the component, choose **Add new workload**.



  - If **Add new workload** doesn't appear, this component doesn't support multiple workloads.
  - If the **Associated workloads** heading doesn't appear, this component doesn't support customized workloads.
- To remove a workload, choose **Remove** next to the workload that you want to remove from monitoring.



- To disable monitoring for the entire component, clear the **Monitoring** check box.



- When you are done editing the component, choose **Save changes** in the lower right corner. Any changes to workloads for a component are visible on the **Review components for monitoring** table under **Associated workloads**.

8. On the **Review detected components** page, choose **Next**.
9. The **Specify component details** page includes all components with customizable associated workloads from the previous step.

> **Note**
>
> If a component header has an *optional* tag, additional details for the workloads in that component are optional.

   If a component doesn't appear on this page, the component doesn't have any additional details that can be specified in this step.

10. Choose **Next**.
11. On the **Review and submit** page, review all monitored component and workload details.

12. Choose **Submit**.

Account-based application monitoring

1. **Application name**. Enter a name for your account-based application.
2. **Automated monitoring of new resources**. By default, Application Insights uses recommended settings to configure monitoring for resource components that are added to your account after you onboard the application. You can exclude monitoring for resources added after onboarding your application by clearing the check box.
3. **Monitor CloudWatch Events**. Select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs And Notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
4. **Integrate with AWS Systems Manager OpsCenter.** To view and get notified when problems are detected for selected applications, select the **Generate Systems Manager OpsCenter OpsItems for remedial actions** check box. To track the operations that are taken to resolve operational work items (OpsItems) that are related to your AWS resources, provide the SNS topic ARN.
5. **Tags — optional**. CloudWatch Application Insights supports both tag-based and CloudFormation-based resource groups (with the exception of Auto Scaling groups). For more information, see Working with Tag Editor.
6. **Discovered resources**. All of the resources discovered in your account are added to this list. If Application Insights is unable to discover all of the resources in your account, an error message appears at the top of the page. This message includes a link to the documentation for how to add the required permissions.
7. Choose **Next**.

   An ARN is generated for the application in the following format.

   ```
   arn:partition:applicationinsights:region:account-id:application/
   TBD/application-name
   ```

   Example

```
arn:aws:applicationinsights:us-east-1:123456789012:application/TBD/my-
application
```

4. After you submit your application monitoring configuration, you will be taken to the details page for the application, where you can view the **Application summary**, the list of **Monitored components** and **Unmonitored components**, and, by selecting the tabs next to **Components**, the **Configuration history**, **Log patterns**, and any **Tags** that you have applied.

   To view insights for the application, choose **View Insights**.

   You can update your selections for CloudWatch Events monitoring and integration with AWS Systems Manager OpsCenter by choosing **Edit**.

   Under **Components**, you can select the **Actions** menu to Create, Modify, or Ungroup an instance group.

   You can manage monitoring for components, including application tier, log groups, event logs, metrics, and custom alarms, by selecting the bullet next to a component and choosing **Manage monitoring**.

Enable Application Insights for Amazon ECS and Amazon EKS resource monitoring

You can enable Application Insights to monitor containerized applications and microservices from the Container Insights console. Application Insights supports monitoring for the following resources:

- Amazon ECS clusters
- Amazon ECS services
- Amazon ECS tasks
- Amazon EKS clusters

When Application Insights is enabled, it provides recommended metrics and logs, detects potential problems, generates CloudWatch Events, and creates automatic dashboards for your containerized applications and microservices.

You can enable Application Insights for containerized resources from the Container Insights or Application Insights consoles.

Enable Application Insights from the Container Insights console

From the Container Insights console, on the Container Insights **Performance monitoring** dashboard, choose **Auto-configure Application Insights**. When Application Insights is enabled, it displays details about detected problems.

Enable Application Insights from the Application Insights console

When ECS clusters appear in the component list, Application Insights automatically enables additional container monitoring with Container Insights.

For EKS clusters, you can enable additional monitoring with Container Insights to provide diagnostics information, such as container restart failures, to help you isolate and resolve problems. Additional steps are required to set up Container Insights for EKS. For information, see Amazon EKS for steps to set up Container Insights on EKS.

Additional monitoring for EKS with Container Insights is supported on Linux instances with EKS.

For more information about Container Insights support for ECS and EKS clusters, see Container Insights.

Disable monitoring for an application component

To disable monitoring for an application component, from the application details page, select the component for which you want to disable monitoring. Choose **Actions**, and then **Remove from monitoring**.

Delete an application

To delete an application, from the CloudWatch dashboard, on the left navigation pane, choose **Application Insights** under **Insights**. Select the application that you want to delete. Under **Actions**, choose **Delete application**. This deletes monitoring and deletes all of the saved monitors for application components. The application resources are not deleted.

Set up, configure, and manage your application for monitoring using the command line

This section provides steps for setting up, configuring, and managing your application for monitoring using the AWS CLI and AWS Tools for Windows PowerShell.

Command line procedures

- Add and manage an application

- Manage and update monitoring
- Configure monitoring for SQL Always On Availability Groups
- Configure monitoring for MySQL RDS
- Configure monitoring for MySQL EC2
- Configure monitoring for PostgreSQL RDS
- Configure monitoring for PostgreSQL EC2
- Configure monitoring for Oracle RDS
- Configure monitoring for Oracle EC2

Add and manage an application

You can add, get information about, manage, and configure your Application Insights application using the command line.

Topics

- Add an application
- Describe an application
- List components in an application
- Describe a component
- Group similar resources into a custom component
- Ungroup a custom component
- Update an application
- Update a custom component

Add an application

Add an application using the AWS CLI

To use the AWS CLI to add an application for your resource group called `my-resource-group`, with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
aws application-insights create-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Add an application using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to add an application for your resource group called `my-resource-group` with OpsCenter enabled to deliver the created opsItem to the SNS topic ARN `arn:aws:sns:us-east-1:123456789012:MyTopic`, use the following command.

```
New-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Describe an application

Describe an application using the AWS CLI

To use the AWS CLI to describe an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights describe-application --resource-group-name my-resource-group
```

Describe an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an application created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIApplication -ResourceGroupName my-resource-group
```

List components in an application

List components in an application using the AWS CLI

To use the AWS CLI to list the components created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-components --resource-group-name my-resource-group
```

List components in an application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list the components created on a resource group called `my-resource-group`, use the following command.

```
Get-CWAIComponentList -ResourceGroupName my-resource-group
```

Describe a component

Describe a component using the AWS CLI

You can use the following AWS CLI command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
aws application-insights describe-component --resource-group-name my-resource-group --
component-name my-component
```

Describe a component using AWS Tools for Windows PowerShell

You can use the following AWS Tools for Windows PowerShell command to describe a component called `my-component` that belongs to an application created on a resource group called `my-resource-group`.

```
Get-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Group similar resources into a custom component

We recommend grouping similar resources, such as .NET web server instances, into custom components for easier on-boarding and better monitoring and insights. CloudWatch Application Insights supports custom groups for EC2 instances.

To group resources into a custom component using the AWS CLI

To use the AWS CLI to group three instances (`arn:aws:ec2:us-east-1:123456789012:instance/i-11111`, `arn:aws:ec2:us-east-1:123456789012:instance/i-22222`, and `arn:aws:ec2:us-east-1:123456789012:instance/i-33333`) together into a custom component called `my-component` for an application created for the resource group called `my-resource-group`, use the following command.

```
aws application-insights create-component --resource-group-name my-
resource-group --component-name my-component --resource-list arn:aws:ec2:us-
east-1:123456789012:instance/i-11111 arn:aws:ec2:us-east-1:123456789012:instance/
i-22222 arn:aws:ec2:us-east-1:123456789012:instance/i-33333
```

To group resources into a custom component using AWS Tools for Windows PowerShell

To use AWS Tools for Windows PowerShell to group three instances (`arn:aws:ec2:us-east-1:123456789012:instance/i-11111`, `arn:aws:ec2:us-east-1:123456789012:instance/i-22222`, and `arn:aws:ec2:us-east-1:123456789012:instance/i-33333`) together into a custom component called `my-component`, for an application created for the resource group called `my-resource-group`, use the following command.

```
New-CWAIComponent -ResourceGroupName my-resource-group -ComponentName my-component
 -ResourceList arn:aws:ec2:us-east-1:123456789012:instance/i-11111,arn:aws:ec2:us-
east-1:123456789012:instance/i-22222,arn:aws:ec2:us-east-1:123456789012:instance/
i-33333
```

Ungroup a custom component

To ungroup a custom component using the AWS CLI

To use the AWS CLI to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
aws application-insights delete-component --resource-group-name my-resource-group --
component-name my-new-component
```

To ungroup a custom component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to ungroup a custom component named `my-component` in an application created on the resource group, `my-resource-group`, use the following command.

```
Remove-CWAIComponent -ComponentName my-component -ResourceGroupName my-resource-group
```

Update an application

Update an application using the AWS CLI

You can use the AWS CLI to update an application to generate AWS Systems Manager OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic`, using the following command.

```
aws application-insights update-application --resource-group-name my-resource-group --
ops-center-enabled --ops-item-sns-topic-arn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update an application using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update an application to generate AWS SSM OpsCenter OpsItems for problems detected with the application, and to associate the created OpsItems to the SNS topic `arn:aws:sns:us-east-1:123456789012:MyTopic` , using the following command.

```
Update-CWAIApplication -ResourceGroupName my-resource-group -OpsCenterEnabled true -
OpsItemSNSTopicArn arn:aws:sns:us-east-1:123456789012:MyTopic
```

Update a custom component

Update a custom component using the AWS CLI

You can use the AWS CLI to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
aws application-insights update-component --resource-group-name my-resource-
group --component-name my-component --new-component-name my-new-component --
resource-list arn:aws:ec2:us-east-1:123456789012:instance/i-44444 arn:aws:ec2:us-
east-1:123456789012:instance/i-55555
```

Update a custom component using AWS Tools for Windows PowerShell

You can use the AWS Tools for Windows PowerShell to update a custom component called `my-component` with a new component name, `my-new-component`, and an updated group of instances, by using the following command.

```
Update-CWAIComponent -ComponentName my-component -NewComponentName my-new-
component -ResourceGroupName my-resource-group -ResourceList arn:aws:ec2:us-
east-1:123456789012:instance/i-44444,arn:aws:ec2:us-east-1:123456789012:instance/
i-55555
```

Manage and update monitoring

You can manage and update monitoring for your Application Insights application using the command line.

Topics

- List problems with your application
- Describe an application problem
- Describe the anomalies or errors associated with a problem
- Describe an anomaly or error with the application
- Describe the monitoring configurations of a component
- Describe the recommended monitoring configuration of a component
- Update the monitoring configurations for a component
- Remove a specified resource group from Application Insights monitoring

List problems with your application

List problems with your application using the AWS CLI

To use the AWS CLI to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
aws application-insights list-problems --resource-group-name my-resource-group --start-time 1000 --end-time 10000
```

List problems with your application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to list problems with your application detected between 1,000 and 10,000 milliseconds since Unix Epoch for an application created on a resource group called `my-resource-group`, use the following command.

```
$startDate = "8/6/2019 3:33:00"
$endDate = "8/6/2019 3:34:00"
Get-CWAIProblemList -ResourceGroupName my-resource-group -StartTime $startDate -EndTime $endDate
```

Describe an application problem

Describe an application problem using the AWS CLI

To use the AWS CLI to describe a problem with problem id `p-1234567890`, use the following command.

```
aws application-insights describe-problem –problem-id p-1234567890
```

Describe an application problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe a problem with problem id p-1234567890, use the following command.

```
Get-CWAIProblem -ProblemId p-1234567890
```

Describe the anomalies or errors associated with a problem

Describe the anomalies or errors associated with a problem using the AWS CLI

To use the AWS CLI to describe the anomalies or errors associated with a problem with problem id p-1234567890, use the following command.

```
aws application-insights describe-problem-observations --problem-id -1234567890
```

Describe the anomalies or errors associated with a problem using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the anomalies or errors associated with a problem with problem id p-1234567890, use the following command.

```
Get-CWAIProblemObservation -ProblemId p-1234567890
```

Describe an anomaly or error with the application

Describe an anomaly or error with the application using the AWS CLI

To use the AWS CLI to describe an anomaly or error with the application with the observation id o-1234567890, use the following command.

```
aws application-insights describe-observation –observation-id o-1234567890
```

Describe an anomaly or error with the application using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe an anomaly or error with the application with the observation id o-1234567890, use the following command.

```
Get-CWAIObservation -ObservationId o-1234567890
```

Describe the monitoring configurations of a component

Describe the monitoring configurations of a component using the AWS CLI

To use the AWS CLI to describe the monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, use the following command.

```
aws application-insights describe-component-configuration –resource-group-name my-resource-group –component-name my-component
```

Describe the monitoring configurations of a component using AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to describe the monitoring configuration of a component called `my-component`, in an application created on the resource group `my-resource-group`, use the following command.

```
Get-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-resource-group
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Describe the recommended monitoring configuration of a component

Describe the recommended monitoring configuration of a component using the AWS CLI

When the component is part of a .NET Worker application, you can use the AWS CLI to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
aws application-insights describe-component-configuration-recommendation --resource-group-name my-resource-group --component-name my-component --tier DOT_NET_WORKER
```

Describe the recommended monitoring configuration of a component using AWS Tools for Windows PowerShell

When the component is part of a .NET Worker application, you can use the AWS Tools for Windows PowerShell to describe the recommended monitoring configuration of a component called `my-component` in an application created on the resource group `my-resource-group`, by using the following command.

```
Get-CWAIComponentConfigurationRecommendation -ComponentName my-component -
ResourceGroupName my-resource-group -Tier DOT_NET_WORKER
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Update the monitoring configurations for a component

Update the monitoring configurations for a component using the AWS CLI

To use the AWS CLI to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.
3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
aws application-insights update-component-configuration --resource-group-name my-
resource-group --component-name my-component --tier DOT_NET_WORKER --monitor --
component-configuration "file://configuration.txt"
```

Update the monitoring configurations for a component using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to update the component called `my-component` in an application created on the resource group called `my-resource-group`, use the following command. The command includes these actions:

1. Enable monitoring for the component.
2. Set the tier of the component to `.NET Worker`.

3. Update the JSON configuration of the component to read from the local file `configuration.txt`.

```
[string]$config = Get-Content -Path configuration.txt
Update-CWAIComponentConfiguration -ComponentName my-component -ResourceGroupName my-
resource-group -Tier DOT_NET_WORKER -Monitor 1 -ComponentConfiguration $config
```

For more information about component configuration and for example JSON files, see Work with component configurations.

Remove a specified resource group from Application Insights monitoring

Remove a specified resource group from Application Insights monitoring using the AWS CLI

To use the AWS CLI to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
aws application-insights delete-application --resource-group-name my-resource-group
```

Remove a specified resource group from Application Insights monitoring using the AWS Tools for Windows PowerShell

To use the AWS Tools for Windows PowerShell to remove an application created on the resource group called `my-resource-group` from monitoring, use the following command.

```
Remove-CWAIApplication -ResourceGroupName my-resource-group
```

Configure monitoring for SQL Always On Availability Groups

1. Create an application for the resource group with the SQL HA EC2 instances.

   ```
   aws application-insights create-application --region <REGION> --resource-group-name
     <RESOURCE_GROUP_NAME>
   ```

2. Define the EC2 instances that represent the SQL HA cluster by creating a new application component.

   ```
   aws application-insights create-component --resource-group-name
    "<RESOURCE_GROUP_NAME>" --component-name SQL_HA_CLUSTER --resource-list
   ```

```
  "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_1_ID>"
 "arn:aws:ec2:<REGION>:<ACCOUNT_ID>:instance/<CLUSTER_INSTANCE_2_ID>
```

3. Configure the SQL HA component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "SQL_HA_CLUSTER" --
monitor --tier SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP --monitor  --component-
configuration '{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/
sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
```

```
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-
<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\
\MSSQL\\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
```

```
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}'
```

> **Note**
>
> Application Insights must ingest Application Event logs (information level) to detect cluster activities such as failover.

Configure monitoring for MySQL RDS

1. Create an application for the resource group with the RDS MySQL database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see Accessing the MySQL Slow Query and General Logs.
   - `set slow_query_log = 1`
   - `set log_output = FILE`
3. Export the logs to be monitored to CloudWatch logs. For more information, see Publishing MySQL Logs to CloudWatch Logs.
4. Configure the MySQL RDS component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier DEFAULT --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logType\":
\"MYSQL\",\"monitor\":true},{\"logType\": \"MYSQL_SLOW_QUERY\",\"monitor\":false}]}"
```

Configure monitoring for MySQL EC2

1. Create an application for the resource group with the SQL HA EC2 instances.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. The error log is enabled by default. The slow query log can be enabled using data parameter groups. For more information, see Accessing the MySQL Slow Query and General Logs.
   - `set slow_query_log = 1`
   - `set log_output = FILE`
3. Configure the MySQL EC2 component.

```
aws application-insights  update-component-configuration --resource-group-name
 "<RESOURCE_GROUP_NAME>" --region <REGION> --component-name "<DB_COMPONENT_NAME>"
 --monitor --tier MYSQL --monitor  --component-configuration "{\"alarmMetrics\":
[{\"alarmMetricName\":\"CPUUtilization\",\"monitor\":true}],\"logs\":[{\"logGroupName
\":\"<UNIQUE_LOG_GROUP_NAME>\",\"logPath\":\"C:\\\\ProgramData\\\\MySQL\\\\MySQL
 Server **\\\\Data\\\\<FILE_NAME>.err\",\"logType\":\"MYSQL\",\"monitor\":true,
\"encoding\":\"utf-8\"}]}"
```

Configure monitoring for PostgreSQL RDS

1. Create an application for the resource group with the PostgreSQL RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Publishing PostgreSQL logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check box labeled **PostgreSQL** log. Choose **Continue** to save this setting.
3. Your PostgreSQL logs are exported to CloudWatch.
4. Configure the PostgreSQL RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
```

```
      }
   ],
   \"logs\":[
      {
         \"logType\": \"POSTGRESQL\",
         \"monitor\": true
      }
   ]
}"
```

Configure monitoring for PostgreSQL EC2

1. Create an application for the resource group with the PostgreSQL EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Configure the PostgreSQL EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier POSTGRESQL --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/var/lib/pgsql/data/log/\",
         \"logType\":\"POSTGRESQL\",
         \"monitor\":true,
         \"encoding\":\"utf-8\"
      }
   ]
}"
```

Configure monitoring for Oracle RDS

1. Create an application for the resource group with the Oracle RDS database instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Publishing Oracle logs to CloudWatch is not enabled by default. To enable monitoring, open the RDS console and select the database to monitor. Choose **Modify** in the upper right corner, and select the check boxes labeled **Alert** log and **Listener** log. Choose **Continue** to save this setting.
3. Your Oracle logs are exported to CloudWatch.
4. Configure the Oracle RDS component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier DEFAULT --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\": \"CPUUtilization\",
         \"monitor\": true
      }
   ],
   \"logs\":[
      {
         \"logType\": \"ORACLE_ALERT\",
         \"monitor\": true
      },
      {
         \"logType\": \"ORACLE_LISTENER\",
         \"monitor\": true
      }
   ]
}"
```

Configure monitoring for Oracle EC2

1. Create an application for the resource group with the Oracle EC2 instance.

```
aws application-insights create-application --region <REGION> --resource-group-name
  <RESOURCE_GROUP_NAME>
```

2. Configure the Oracle EC2 component.

```
aws application-insights update-component-configuration --region <REGION> --resource-
group-name <RESOURCE_GROUP_NAME> --component-name <DB_COMPONENT_NAME> --monitor --
tier ORACLE --component-configuration
"{
   \"alarmMetrics\":[
      {
         \"alarmMetricName\":\"CPUUtilization\",
         \"monitor\":true
      }
   ],
   \"logs\":[
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/rdbms/*/*/trace\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      },
      {
         \"logGroupName\":\"<UNIQUE_LOG_GROUP_NAME>\",
         \"logPath\":\"/opt/oracle/diag/tnslsnr/$HOSTNAME/listener/trace/\",
         \"logType\":\"ORACLE_ALERT\",
         \"monitor\":true,
      }
   ]
}"
```

Application Insights CloudWatch Events for detected problems

For each application that is added to CloudWatch Application Insights, a CloudWatch event is published for the following events on a best effort basis:

- **Problem creation.** Emitted when CloudWatch Application Insights detects a new problem.
  - Detail Type: **"Application Insights Problem Detected"**
  - Detail:
    - `problemId`: The detected problem ID.

- `region`: The AWS Region where the problem was created.
- `resourceGroupName`: The Resource Group for the registered application for which the problem was detected.
- `status`: The status of the problem. Possible status and definitions are as follows:
  - `In progress`: A new problem has been identified. The problem is still receiving observations.
  - `Recovering`: The problem is stabilizing. You can manually resolve the problem when it is in this state.
  - `Resolved`: The problem is resolved. There are no new observations about this problem.
  - `Recurring`: The problem was resolved within the past 24 hours. It has reopened as a result of additional observations.
- `severity`: The severity of the problem.
- `problemUrl`: The console URL for the problem.

- **Problem update.** Emitted when the problem is updated with a new observation or when an existing observation is updated and the problem is subsequently updated; updates include a resolution or closure of the problem.
  - Detail Type: **"Application Insights Problem Updated"**
  - Detail:
    - `problemId`: The created problem ID.
    - `region`: The AWS Region where the problem was created.
    - `resourceGroupName`: The Resource Group for the registered application for which the problem was detected.
    - `status`: The status of the problem.
    - `severity`: The severity of the problem.
    - `problemUrl`: The console URL for the problem.

Receive notifications for detected problems

You can use Amazon SNS notifications, Systems Manager OpsCenter, or CloudWatch Events to receive notifications about problems that are detected in your applications.

CloudWatch Application Insights Amazon SNS notifications for detected problems

You can configure Amazon SNS notifications using the AWS Management Console or the AWS CLI. To set up notifications using the AWS Management Console, you must have the necessary Amazon SNS permissions as shown in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "sns:ListTopics",
          "sns:Subscribe",
          "sns:CreateTopic"
        ],
        "Resource": [
          "*"
        ]
      }
    ]
}
```

After you set up Amazon SNS notifications, you receive email notifications when a problem is created or resolved. You also receive notifications when a new observation is added to an existing problem.

The following example shows the content of an email notification.

```
    You are receiving this email because Problem "p-1234567" has been CREATED by Amazon
 CloudWatch Application Insights

    Problem Details:
    Problem URL: https:////console.aws.amazon.com/cloudwatch/home?region=us-
east-1#settings:AppInsightsSettings/problemDetails?problemId=p-1234567
    Problem Summary: Title of the problem
    Severity: HIGH
    Insights: Something specific is broken
    Status : RESOLVED
    AffectedResource: arn:aws:ec2:us-east-1:555555555555:host/testResource
```

```
    Region: us-east-1
    RecurringCount: 0
    StartTime: 2019-03-23T10:42:57.777Z
    LastUpdatedTime: 2019-03-23T21:49:37.777Z
    LastRecurrenceTime:
    StopTime: 2019-03-23T21:49:37.777Z

    Recent Issues
    - TelemetryArn:alarm1
      StartTime: 2024-08-15T22:12:46.007Z
      StopTime:
    - TelemetryArn:log-group1
      StartTime: 2024-08-15T22:12:46.007Z
      StopTime: 2024-08-15T22:12:46.007Z
```

How to receive problem notifications using Systems Manager

Actions through AWS Systems Manager. CloudWatch Application Insights provides built-in integration with Systems Manager OpsCenter. If you choose to use this integration for your application, an OpsItem is created on the OpsCenter console for every problem detected with the application. From the OpsCenter console, you can view summarized information about the problem detected by CloudWatch Application Insights and pick a Systems Manager Automation runbook to take remedial actions or further identify Windows processes that are causing resource issues in your application.

How to receive problem notifications using CloudWatch Events

From the CloudWatch console, select **Rules** under **Events** in the left navigation pane. From the **Rules** page, select **Create rule**. Choose **Amazon CloudWatch Application Insights** from the **Service Name** dropdown list and choose the **Event Type**. Then, choose **Add target** and select the target and parameters, for example, an **SNS topic** or **Lambda function**.

Application Insights cross-account observability

With CloudWatch Application Insights cross-account observability, you can monitor and troubleshoot your applications that span multiple AWS accounts within a single Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your AWS accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a sink in your monitoring account. You use the sink

to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, make sure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Applications in CloudWatch Application Insights
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs
- Traces in AWS X-Ray

Work with component configurations

A component configuration is a text file in JSON format that describes the configuration settings of the component. This section provides an example template fragment, descriptions of component configuration sections, and example component configurations.

Topics

- Component configuration template fragment
- Component configuration sections
- Component configuration examples

Component configuration template fragment

The following example shows a template fragment in JSON format.

```
{
  "alarmMetrics" : [
    list of alarm metrics
  ],
  "logs" : [
    list of logs
  ],
  "processes" : [
   list of processes
```

```
  ],
  "windowsEvents" : [
    list of windows events channels configurations
  ],
  "alarms" : [
    list of CloudWatch alarms
  ],
  "jmxPrometheusExporter": {
    JMX Prometheus Exporter configuration
  },
  "hanaPrometheusExporter": {
      SAP HANA Prometheus Exporter configuration
  },
  "haClusterPrometheusExporter": {
      HA Cluster Prometheus Exporter configuration
  },
  "netWeaverPrometheusExporter": {
      SAP NetWeaver Prometheus Exporter configuration
  },
  "subComponents" : [
    {
      "subComponentType" : "AWS::EC2::Instance" ...
      component nested instances configuration
    },
    {
      "subComponentType" : "AWS::EC2::Volume" ...
      component nested volumes configuration
    }
  ]
}
```

Component configuration sections

A component configuration includes several major sections. Sections in a component configuration can be listed in any order.

- **alarmMetrics (optional)**

  A list of metrics to monitor for the component. All component types can have an alarmMetrics section.

- **logs (optional)**

  A list of logs to monitor for the component. Only EC2 instances can have a logs section.

- **processes (optional)**

  A list of processes to monitor for the component. Only EC2 instances can have a processes section.

- **subComponents (optional)**

  Nested instance and volume subComponent configuration for the component. The following types of components can have nested instances and a subComponents section: ELB, ASG, custom-grouped EC2 instances , and EC2 instances.

- **alarms (optional)**

  A list of alarms to monitor for the component. All component types can have an alarm section.

- **windowsEvents (optional)**

  A list of windows events to monitor for the component. Only Windows on EC2 instances have a `windowsEvents` section.

- **JMXPrometheusExporter (optional)**

  JMXPrometheus Exporter configuration.

- **hanaPrometheusExporter (optional)**

  SAP HANA Prometheus Exporter configuration.

- **haClusterPrometheusExporter (optional)**

  HA Cluster Prometheus Exporter configuration.

- **netWeaverPrometheusExporter (optional)**

  SAP NetWeaver Prometheus Exporter configuration.

- **sapAsePrometheusExporter (optional)**

  SAP ASE Prometheus Exporter configuration.

The following example shows the syntax for the **subComponents section fragment** in JSON format.

```
[
  {
    "subComponentType" : "AWS::EC2::Instance",
```

```
    "alarmMetrics" : [
      list of alarm metrics
    ],
    "logs" : [
      list of logs
    ],
    "processes": [
      list of processes
    ],
    "windowsEvents" : [
      list of windows events channels configurations
    ]
  },
  {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [
      list of alarm metrics
    ]
  }
]
```

Component configuration section properties

This section describes the properties of each component configuration section.

Sections

- Metric
- Log
- Process
- JMX Prometheus Exporter
- HANA Prometheus Exporter
- HA Cluster Prometheus Exporter
- NetWeaver Prometheus Exporter
- SAP ASE Prometheus Exporter
- Windows Events
- Alarm

Metric

Defines a metric to be monitored for the component.

JSON

```
{
  "alarmMetricName" : "monitoredMetricName",
  "monitor" : true/false
}
```

Properties

- **alarmMetricName (required)**

  The name of the metric to be monitored for the component. For metrics supported by Application Insights, see Logs and metrics supported by Amazon CloudWatch Application Insights.

- **monitor (optional)**

  Boolean to indicate whether to monitor the metric. The default value is `true`.

Log

Defines a log to be monitored for the component.

JSON

```
{
  "logGroupName" : "logGroupName",
  "logPath" : "logPath",
  "logType" : "logType",
  "encoding" : "encodingType",
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see CreateLogGroup.

- **logPath (required for EC2 instance components; not required for components that do not use CloudWatch Agent, such as AWS Lambda)**

  The path of the logs to be monitored. The log path must be an absolute Windows system file path. For more information, see CloudWatch Agent Configuration File: Logs Section.

- **logType (required)**

  The log type decides the log patterns against which Application Insights analyzes the log. The log type is selected from the following:

  - `SQL_SERVER`
  - `MYSQL`
  - `MYSQL_SLOW_QUERY`
  - `POSTGRESQL`
  - `ORACLE_ALERT`
  - `ORACLE_LISTENER`
  - `IIS`
  - `APPLICATION`
  - `WINDOWS_EVENTS`
  - `WINDOWS_EVENTS_ACTIVE_DIRECTORY`
  - `WINDOWS_EVENTS_DNS`
  - `WINDOWS_EVENTS_IIS`
  - `WINDOWS_EVENTS_SHAREPOINT`
  - `SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP`
  - `SQL_SERVER_FAILOVER_CLUSTER_INSTANCE`
  - `DEFAULT`
  - `CUSTOM`
  - `STEP_FUNCTION`
  - `API_GATEWAY_ACCESS`
  - `API_GATEWAY_EXECUTION`
  - `SAP_HANA_LOGS`

- `SAP_HANA_TRACE`
- `SAP_HANA_HIGH_AVAILABILITY`
- `SAP_NETWEAVER_DEV_TRACE_LOGS`
- `PACEMAKER_HIGH_AVAILABILITY`

- **encoding (optional)**

  The type of encoding of the logs to be monitored. The specified encoding should be included in the list of CloudWatch agent supported encodings. If not provided, CloudWatch Application Insights uses the default encoding of type utf-8, except for:

  - `SQL_SERVER`: utf-16 encoding
  - `IIS`: ascii encoding

- **monitor (optional)**

  Boolean that indicates whether to monitor the logs. The default value is `true`.

Process

Defines a process to be monitored for the component.

JSON

```
{
  "processName" : "monitoredProcessName",
  "alarmMetrics" : [
      list of alarm metrics
  ]
}
```

Properties

- **processName (required)**

  The name of the process to be monitored for the component. The process name must not contain a process stem, such as `sqlservr` or `sqlservr.exe`.

- **alarmMetrics (required)**

  A list of metrics to monitor for this process. To view process metrics supported by CloudWatch Application Insights, see Amazon Elastic Compute Cloud (EC2).

JMX Prometheus Exporter

Defines the JMX Prometheus Exporter settings.

JSON

```
"JMXPrometheusExporter": {
  "jmxURL" : "JMX URL",
  "hostPort" : "The host and port",
  "prometheusPort" : "Target port to emit Prometheus metrics"
}
```

Properties

- **jmxURL (optional)**

  A complete JMX URL to connect to.
- **hostPort (optional)**

  The host and port to connect to through remote JMX. Only one of `jmxURL` and `hostPort` can be specified.
- **prometheusPort (optional)**

  The target port to send Prometheus metrics to. If not specified, the default port 9404 is used.

HANA Prometheus Exporter

Defines the HANA Prometheus Exporter settings.

JSON

```
"hanaPrometheusExporter": {
    "hanaSid": "SAP HANA  SID",
    "hanaPort": "HANA database port",
    "hanaSecretName": "HANA secret name",
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **hanaSid**

The three-character SAP system ID (SID) of the SAP HANA system.

- **hanaPort**

  The HANA database port by which the exporter will query HANA metrics.

- **hanaSecretName**

  The AWS Secrets Manager secret that stores HANA monitoring user credentials. The HANA Prometheus exporter uses these credentials to connect to the database and query HANA metrics.

- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9668 is used.

HA Cluster Prometheus Exporter

Defines the HA Cluster Prometheus Exporter settings.

JSON

```
"haClusterPrometheusExporter": {
    "prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9664 is used.

NetWeaver Prometheus Exporter

Defines the NetWeaver Prometheus Exporter settings.

JSON

```
"netWeaverPrometheusExporter": {
    "sapSid": "SAP NetWeaver  SID",
    "instanceNumbers": [ "Array of instance Numbers of SAP NetWeaver system "],
```

```
"prometheusPort": "Target port to emit Prometheus metrics"
}
```

Properties

- **sapSid**

  The 3 character SAP system ID (SID) of the SAP NetWeaver system.
- **instanceNumbers**

  Array of the instance Numbers of SAP NetWeaver system.

  Example: `"instanceNumbers": [ "00", "01"]`
- **prometheusPort (optional)**

  The target port to which to send Prometheus metrics. If not specified, the default port `9680` is used.

SAP ASE Prometheus Exporter

Defines the SAP ASE Prometheus Exporter settings.

JSON

```
"sapASEPrometheusExporter": {
    "sapAseSid": "SAP ASE SID",
    "sapAsePort": "SAP ASE database port",
    "sapAseSecretName": "SAP ASE secret name",
    "prometheusPort": "Target port to emit Prometheus metrics",
    "agreeToEnableASEMonitoring": true
}
```

Properties

- **sapAseSid**

  The three-character SAP system ID (SID) of the SAP ASE system.
- **sapAsePort**

  The SAP ASE database port by which the exporter will query ASE metrics.
- **sapAseSecretName**

The AWS Secrets Manager secret that stores ASE monitoring user credentials. The SAP ASE Prometheus exporter uses these credentials to connect to the database and query ASE metrics.

- **prometheusPort (optional)**

  The target port to which Prometheus sends metrics. If not specified, the default port 9399 is used. If there is another ASE DB that is using the default port, then we use 9499.

Windows Events

Defines Windows Events to log.

JSON

```
{
  "logGroupName" : "logGroupName",
  "eventName" : "eventName",
  "eventLevels" : ["ERROR","WARNING","CRITICAL","INFORMATION","VERBOSE"],
  "monitor" : true/false
}
```

Properties

- **logGroupName (required)**

  The CloudWatch log group name to be associated to the monitored log. For the log group name constraints, see CreateLogGroup.

- **eventName (required)**

  The type of Windows Events to log. It is equivalent to the Windows Event log channel name. For example, System, Security, CustomEventName, and so on. This field is required for each type of Windows event to log.

- **eventLevels (required)**

  The levels of event to log. You must specify each level to log. Possible values include `INFORMATION`, `WARNING`, `ERROR`, `CRITICAL`, and `VERBOSE`. This field is required for each type of Windows Event to log.

- **monitor (optional)**

  Boolean that indicates whether to monitor the logs. The default value is `true`.

Alarm

Defines a CloudWatch alarm to be monitored for the component.

JSON

```
{
  "alarmName" : "monitoredAlarmName",
  "severity" : HIGH/MEDIUM/LOW
}
```

Properties

- **alarmName (required)**

  The name of the CloudWatch alarm to be monitored for the component.
- **severity (optional)**

  Indicates the degree of outage when the alarm goes off.

Component configuration examples

The following examples show component configurations in JSON format for relevant services.

Example component configurations

- Amazon DynamoDB table
- Amazon EC2 Auto Scaling (ASG)
- Amazon EKS cluster
- Amazon Elastic Compute Cloud (EC2) instance
- Amazon Elastic Container Service (Amazon ECS)
- Amazon ECS services
- Amazon ECS tasks
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx
- Amazon Relational Database Service (RDS) Aurora MySQL
- Amazon Relational Database Service (RDS) instance

- Amazon Route 53 health check
- Amazon Route 53 hosted zone
- Amazon Route 53 Resolver endpoint
- Amazon Route 53 Resolver query logging configuration
- Amazon S3 bucket
- Amazon Simple Queue Service (SQS)
- Amazon SNS topic
- Amazon Virtual Private Cloud (Amazon VPC)
- Amazon VPC Network Address Translation (NAT) gateways
- API Gateway REST API stages
- Application Elastic Load Balancing
- AWS Lambda Function
- AWS Network Firewall rule group
- AWS Network Firewall rule group association
- AWS Step Functions
- Customer-grouped Amazon EC2 instances
- Elastic Load Balancing
- Java
- Kubernetes on Amazon EC2
- RDS MariaDB and RDS MySQL
- RDS Oracle
- RDS PostgreSQL
- SAP ASE on Amazon EC2
- SAP ASE High Availability on Amazon EC2
- SAP HANA on Amazon EC2
- SAP HANA High Availability on Amazon EC2
- SAP NetWeaver on Amazon EC2
- SAP NetWeaver High Availability on Amazon EC2
- SQL Always On Availability Group

- SQL failover cluster instance

Amazon DynamoDB table

The following example shows a component configuration in JSON format for Amazon DynamoDB table.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "SystemErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "UserErrors",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedReadCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ConsumedWriteCapacityUnits",
      "monitor": false
    },
    {
      "alarmMetricName": "ReadThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "WriteThrottleEvents",
      "monitor": false
    },
    {
      "alarmMetricName": "ConditionalCheckFailedRequests",
      "monitor": false
    },
    {
      "alarmMetricName": "TransactionConflict",
      "monitor": false
    }
  ],
  "logs": []
```

```
}
```

Amazon EC2 Auto Scaling (ASG)

The following example shows a component configuration in JSON format for Amazon EC2 Auto Scaling (ASG).

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUCreditBalance"
      }, {
        "alarmMetricName" : "EBSIOBalance%"
      }
    ],
    "subComponents" : [
      {
        "subComponentType" : "AWS::EC2::Instance",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "CPUUtilization"
          }, {
            "alarmMetricName" : "StatusCheckFailed"
          }
        ],
        "logs" : [
          {
            "logGroupName" : "my_log_group",
            "logPath" : "C:\\LogFolder\\*",
            "logType" : "APPLICATION"
          }
        ],
        "processes" : [
          {
            "processName" : "my_process",
            "alarmMetrics" : [
              {
                  "alarmMetricName" : "procstat cpu_usage",
                  "monitor" : true
              }, {
                  "alarmMetricName" : "procstat memory_rss",
                  "monitor" : true
              }
```

```
          ]
      }
  ],
        "windowsEvents" : [
          {
            "logGroupName" : "my_log_group_2",
            "eventName" : "Application",
            "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType" : "AWS::EC2::Volume",
        "alarmMetrics" : [
          {
            "alarmMetricName" : "VolumeQueueLength"
          }, {
            "alarmMetricName" : "BurstBalance"
          }
        ]
      }
    ],
    "alarms" : [
      {
        "alarmName" : "my_asg_alarm",
        "severity" : "LOW"
      }
    ]
  }
```

Amazon EKS cluster

The following example shows a component configuration in JSON format for Amazon EKS cluster.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName": "cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName": "node_cpu_reserved_capacity",
          "monitor":true
       },
       {
```

```
            "alarmMetricName": "node_cpu_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "node_filesystem_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "node_memory_reserved_capacity",
            "monitor":true
        },
        {
            "alarmMetricName": "node_memory_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "node_network_total_bytes",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_cpu_reserved_capacity",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_cpu_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_cpu_utilization_over_pod_limit",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_reserved_capacity",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_utilization",
            "monitor":true
        },
        {
            "alarmMetricName": "pod_memory_utilization_over_pod_limit",
            "monitor":true
        },
        {
```

```
            "alarmMetricName": "pod_network_rx_bytes",
            "monitor":true
         },
         {
            "alarmMetricName": "pod_network_tx_bytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName": "/aws/containerinsights/kubernetes/application",
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::EC2::Instance",
            "alarmMetrics":[
               {
                  "alarmMetricName":"CPUUtilization",
                  "monitor":true
               },
               {
                  "alarmMetricName":"StatusCheckFailed",
                  "monitor":true
               },
               {
                  "alarmMetricName":"disk_used_percent",
                  "monitor":true
               },
               {
                  "alarmMetricName":"mem_used_percent",
                  "monitor":true
               }
            ],
            "logs":[
               {
                  "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                  "logPath":"",
                  "logType":"APPLICATION",
                  "monitor":true,
                  "encoding":"utf-8"
```

```
            }
          ],
          "processes" : [
            {
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                ]
            }
        ],
          "windowsEvents":[
             {
                "logGroupName":"my_log_group_2",
                "eventName":"Application",
                "eventLevels":[
                   "ERROR",
                   "WARNING",
                   "CRITICAL"
                ],
                "monitor":true
             }
          ]
       },
       {
          "subComponentType":"AWS::AutoScaling::AutoScalingGroup",
          "alarmMetrics":[
             {
                "alarmMetricName":"CPUCreditBalance",
                "monitor":true
             },
             {
                "alarmMetricName":"EBSIOBalance%",
                "monitor":true
             }
          ]
       },
       {
          "subComponentType":"AWS::EC2::Volume",
```

```
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeReadBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteBytes",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeReadOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeWriteOps",
               "monitor":true
            },
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":true
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":true
            }
         ]
      }
   ]
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance`, `AWS::EC2::Volume`, and `AWS::AutoScaling::AutoScalingGroup` applies only to Amazon EKS cluster running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon Elastic Compute Cloud (EC2) instance

The following example shows a component configuration in JSON format for an Amazon EC2 instance.

> **Important**
>
> When an Amazon EC2 instance enters a `stopped` state, it is removed from monitoring. When it returns to a `running` state, it is added to the list of **Unmonitored components** on the **Application details** page of the CloudWatch Application Insights console. If automatic monitoring of new resources is enabled for the application, the instance is added to the list of **Monitored components**. However, the logs and metrics are set to the default for the workload. The previous log and metrics configuration is not saved.

```
{
    "alarmMetrics" : [
      {
        "alarmMetricName" : "CPUUtilization",
        "monitor" : true
      }, {
        "alarmMetricName" : "StatusCheckFailed"
      }
    ],
    "logs" : [
      {
        "logGroupName" : "my_log_group",
        "logPath" : "C:\\LogFolder\\*",
        "logType" : "APPLICATION",
        "monitor" : true
      },
      {
        "logGroupName" : "my_log_group_2",
        "logPath" : "C:\\LogFolder2\\*",
        "logType" : "IIS",
        "encoding" : "utf-8"
      }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
```

```
                {
                    "alarmMetricName" : "procstat cpu_usage",
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ],
    "windowsEvents" : [
      {
        "logGroupName" : "my_log_group_3",
        "eventName" : "Application",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
      }, {
        "logGroupName" : "my_log_group_4",
        "eventName" : "System",
        "eventLevels" : [ "ERROR", "WARNING", "CRITICAL" ],
        "monitor" : true
    }],
     "alarms" : [
      {
        "alarmName" : "my_instance_alarm_1",
        "severity" : "HIGH"
      },
      {
        "alarmName" : "my_instance_alarm_2",
        "severity" : "LOW"
      }
    ],
     "subComponents" : [
     {
       "subComponentType" : "AWS::EC2::Volume",
       "alarmMetrics" : [
       {
         "alarmMetricName" : "VolumeQueueLength",
         "monitor" : "true"
       },
       {
         "alarmMetricName" : "VolumeThroughputPercentage",
         "monitor" : "true"
       },
```

```
      {
        "alarmMetricName" : "BurstBalance",
        "monitor" : "true"
      }]
    }]
  }
```

Amazon Elastic Container Service (Amazon ECS)

The following example shows a component configuration in JSON format for Amazon Elastic Container Service (Amazon ECS).

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkRxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkTxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"RunningTaskCount",
          "monitor":true
       },
       {
          "alarmMetricName":"PendingTaskCount",
          "monitor":true
       },
       {
          "alarmMetricName":"StorageReadBytes",
          "monitor":true
       },
       {
```

```
            "alarmMetricName":"StorageWriteBytes",
            "monitor":true
         }
      ],
      "logs":[
         {
            "logGroupName":"/ecs/my-task-definition",
            "logType":"APPLICATION",
            "monitor":true
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Backend_4XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"HTTPCode_Backend_5XX",
                  "monitor":true
               },
               {
                  "alarmMetricName":"Latency",
                  "monitor":true
               },
               {
                  "alarmMetricName":"SurgeQueueLength",
                  "monitor":true
               },
               {
                  "alarmMetricName":"UnHealthyHostCount",
                  "monitor":true
               }
            ]
         },
         {
            "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
            "alarmMetrics":[
               {
                  "alarmMetricName":"HTTPCode_Target_4XX_Count",
                  "monitor":true
               },
```

```
            {
               "alarmMetricName":"HTTPCode_Target_5XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"TargetResponseTime",
               "monitor":true
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Instance",
         "alarmMetrics":[
            {
               "alarmMetricName":"CPUUtilization",
               "monitor":true
            },
            {
               "alarmMetricName":"StatusCheckFailed",
               "monitor":true
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
```

```
                "processName" : "my_process",
                "alarmMetrics" : [
                    {
                        "alarmMetricName" : "procstat cpu_usage",
                        "monitor" : true
                    }, {
                        "alarmMetricName" : "procstat memory_rss",
                        "monitor" : true
                    }
                 ]
            }
         ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
               "monitor":"true"
            },
            {
               "alarmMetricName":"VolumeThroughputPercentage",
               "monitor":"true"
            },
            {
               "alarmMetricName":"BurstBalance",
               "monitor":"true"
            }
         ]
      }
   ]
```

```
}
```

Note

- The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS clusters with ECS service or ECS task running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS services

The following example shows a component configuration in JSON format for an Amazon ECS service.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"CPUUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilization",
          "monitor":true
       },
       {
          "alarmMetricName":"CpuUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"MemoryUtilized",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkRxBytes",
          "monitor":true
       },
       {
          "alarmMetricName":"NetworkTxBytes",
          "monitor":true
```

```
        },
        {
           "alarmMetricName":"RunningTaskCount",
           "monitor":true
        },
        {
           "alarmMetricName":"PendingTaskCount",
           "monitor":true
        },
        {
           "alarmMetricName":"StorageReadBytes",
           "monitor":true
        },
        {
           "alarmMetricName":"StorageWriteBytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName":"/ecs/my-task-definition",
           "logType":"APPLICATION",
           "monitor":true
        }
     ],
     "subComponents":[
        {
           "subComponentType":"AWS::ElasticLoadBalancing::LoadBalancer",
           "alarmMetrics":[
              {
                 "alarmMetricName":"HTTPCode_Backend_4XX",
                 "monitor":true
              },
              {
                 "alarmMetricName":"HTTPCode_Backend_5XX",
                 "monitor":true
              },
              {
                 "alarmMetricName":"Latency",
                 "monitor":true
              },
              {
                 "alarmMetricName":"SurgeQueueLength",
                 "monitor":true
```

```
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::ElasticLoadBalancingV2::LoadBalancer",
         "alarmMetrics":[
            {
               "alarmMetricName":"HTTPCode_Target_4XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"HTTPCode_Target_5XX_Count",
               "monitor":true
            },
            {
               "alarmMetricName":"TargetResponseTime",
               "monitor":true
            },
            {
               "alarmMetricName":"UnHealthyHostCount",
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Instance",
         "alarmMetrics":[
            {
               "alarmMetricName":"CPUUtilization",
               "monitor":true
            },
            {
               "alarmMetricName":"StatusCheckFailed",
               "monitor":true
            },
            {
               "alarmMetricName":"disk_used_percent",
               "monitor":true
            },
            {
```

```
               "alarmMetricName":"mem_used_percent",
               "monitor":true
            }
         ],
         "logs":[
            {
               "logGroupName":"my_log_group",
               "logPath":"/mylog/path",
               "logType":"APPLICATION",
               "monitor":true
            }
         ],
         "processes" : [
            {
               "processName" : "my_process",
               "alarmMetrics" : [
               {
                   "alarmMetricName" : "procstat cpu_usage",
                   "monitor" : true
               }, {
                   "alarmMetricName" : "procstat memory_rss",
                   "monitor" : true
               }
           ]
       }
   ],
         "windowsEvents":[
            {
               "logGroupName":"my_log_group_2",
               "eventName":"Application",
               "eventLevels":[
                  "ERROR",
                  "WARNING",
                  "CRITICAL"
               ],
               "monitor":true
            }
         ]
      },
      {
         "subComponentType":"AWS::EC2::Volume",
         "alarmMetrics":[
            {
               "alarmMetricName":"VolumeQueueLength",
```

```
                "monitor":"true"
             },
             {
                "alarmMetricName":"VolumeThroughputPercentage",
                "monitor":"true"
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":"true"
             }
          ]
       }
    ]
 }
```

Note

- The `subComponents` section of `AWS::EC2::Instance` and `AWS::EC2::Volume` applies only to Amazon ECS running on the EC2 launch type.
- The `windowsEvents` section of `AWS::EC2::Instance` in `subComponents` applies only to Windows running on Amazon EC2 instances.

Amazon ECS tasks

The following example shows a component configuration in JSON format for an Amazon ECS task.

```
{
    "logs":[
       {
          "logGroupName":"/ecs/my-task-definition",
          "logType":"APPLICATION",
          "monitor":true
       }
    ],
    "processes" : [
        {
            "processName" : "my_process",
            "alarmMetrics" : [
                {
                    "alarmMetricName" : "procstat cpu_usage",
```

```
                    "monitor" : true
                }, {
                    "alarmMetricName" : "procstat memory_rss",
                    "monitor" : true
                }
            ]
        }
    ]
}
```

Amazon Elastic File System (Amazon EFS)

The following example shows a component configuration in JSON format for Amazon EFS.

```
{
   "alarmMetrics": [
     {
       "alarmMetricName": "BurstCreditBalance",
       "monitor": true
     },
     {
       "alarmMetricName": "PercentIOLimit",
       "monitor": true
     },
     {
       "alarmMetricName": "PermittedThroughput",
       "monitor": true
     },
     {
       "alarmMetricName": "MeteredIOBytes",
       "monitor": true
     },
     {
       "alarmMetricName": "TotalIOBytes",
       "monitor": true
     },
     {
       "alarmMetricName": "DataWriteIOBytes",
       "monitor": true
     },
     {
       "alarmMetricName": "DataReadIOBytes",
       "monitor": true
     },
```

```
    {
      "alarmMetricName": "MetadataIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "ClientConnections",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeSinceLastSync",
      "monitor": true
    },
    {
      "alarmMetricName": "Throughput",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfPermittedThroughputUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "ThroughputIOPS",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentThroughputDataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfIOPSDataReadIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "PercentageOfIOPSDataWriteIOBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "AverageDataReadIOBytesSize",
      "monitor": true
    },
```

```
     {
       "alarmMetricName": "AverageDataWriteIOBytesSize",
       "monitor": true
     }
   ],
   "logs": [
    {
    "logGroupName": "/aws/efs/utils",
    "logType": "EFS_MOUNT_STATUS",
    "monitor": true,
    }
   ]
 }
```

Amazon FSx

The following example shows a component configuration in JSON format for Amazon FSx.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DataReadBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteBytes",
      "monitor": true
    },
    {
      "alarmMetricName": "DataReadOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "DataWriteOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "MetadataOperations",
      "monitor": true
    },
    {
      "alarmMetricName": "FreeStorageCapacity",
      "monitor": true
    }
```

```
  ]
}
```

Amazon Relational Database Service (RDS) Aurora MySQL

The following example shows a component configuration in JSON format for Amazon RDS Aurora MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    },
    {
      "alarmMetricName": "CommitLatency",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

Amazon Relational Database Service (RDS) instance

The following example shows a component configuration in JSON format for an Amazon RDS instance.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "BurstBalance",
      "monitor" : true
    }, {
      "alarmMetricName" : "WriteThroughput",
```

```
      "monitor" : false
    }
  ],

  "alarms" : [
    {
      "alarmName" : "my_rds_instance_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon Route 53 health check

The following example shows a component configuration in JSON format for Amazon Route 53 health check.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ChildHealthCheckHealthyCount",
      "monitor": true
    },
    {
      "alarmMetricName": "ConnectionTime",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckPercentageHealthy",
      "monitor": true
    },
    {
      "alarmMetricName": "HealthCheckStatus",
      "monitor": true
    },
    {
      "alarmMetricName": "SSLHandshakeTime",
      "monitor": true
    },
    {
      "alarmMetricName": "TimeToFirstByte",
      "monitor": true
    }
```

```
  ]
}
```

Amazon Route 53 hosted zone

The following example shows a component configuration in JSON format for Amazon Route 53 hosted zone.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "DNSQueries",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECInternalFailure",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeysNeedingAction",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyMaxNeedingActionAge",
      "monitor": true
    },
    {
      "alarmMetricName": "DNSSECKeySigningKeyAge",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logGroupName":"/hosted-zone/logs",
      "logType": "ROUTE53_DNS_PUBLIC_QUERY_LOGS",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver endpoint

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver endpoint.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "EndpointHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "EndpointUnHealthyENICount",
      "monitor": true
    },
    {
      "alarmMetricName": "InboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryVolume",
      "monitor": true
    },
    {
      "alarmMetricName": "OutboundQueryAggregateVolume",
      "monitor": true
    }
  ]
}
```

Amazon Route 53 Resolver query logging configuration

The following example shows a component configuration in JSON format for Amazon Route 53 Resolver query logging configuration.

```
{
  "logs": [
    {
      "logGroupName": "/resolver-query-log-config/logs",
      "logType": "ROUTE53_RESOLVER_QUERY_LOGS",
      "monitor": true
    }
  ]
```

```
}
```

Amazon S3 bucket

The following example shows a component configurations in JSON format for Amazon S3 bucket.

```
{
    "alarmMetrics" : [
        {
            "alarmMetricName" : "ReplicationLatency",
            "monitor" : true
        }, {
            "alarmMetricName" : "5xxErrors",
            "monitor" : true
        }, {
            "alarmMetricName" : "BytesDownloaded"
            "monitor" : true
        }
    ]
}
```

Amazon Simple Queue Service (SQS)

The following example shows a component configuration in JSON format for Amazon Simple Queue Service.

```
{
  "alarmMetrics" : [
    {
      "alarmMetricName" : "ApproximateAgeOfOldestMessage"
    }, {
      "alarmMetricName" : "NumberOfEmptyReceives"
    }
  ],
  "alarms" : [
    {
      "alarmName" : "my_sqs_alarm",
      "severity" : "MEDIUM"
    }
  ]
}
```

Amazon SNS topic

The following example shows a component configuration in JSON format for Amazon SNS topic.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NumberOfNotificationsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-InvalidAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFilteredOut-NoMessageAttributes",
      "monitor": true
    },
    {
      "alarmMetricName": "NumberOfNotificationsFailedToRedriveToDlq",
      "monitor": true
    }
  ]
}
```

Amazon Virtual Private Cloud (Amazon VPC)

The following example shows a component configuration in JSON format for Amazon VPC.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "NetworkAddressUsage",
      "monitor": true
    },
    {
      "alarmMetricName": "NetworkAddressUsagePeered",
      "monitor": true
    },
    {
      "alarmMetricName": "VPCFirewallQueryVolume",
      "monitor": true
    }
```

```
  ]
}
```

Amazon VPC Network Address Translation (NAT) gateways

The following example shows a component configuration in JSON format for NAT gateways.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ErrorPortAllocation",
      "monitor": true
    },
    {
      "alarmMetricName": "IdleTimeoutCount",
      "monitor": true
    }
  ]
}
```

API Gateway REST API stages

The following example shows a component configuration in JSON format for API Gateway REST API stages.

```
{
     "alarmMetrics" : [
         {
             "alarmMetricName" : "4XXError",
             "monitor" : true
         },
         {
             "alarmMetricName" : "5XXError",
             "monitor" : true
         }
     ],
    "logs" : [
        {
            "logType" : "API_GATEWAY_EXECUTION",
            "monitor" : true
        },
        {
            "logType" : "API_GATEWAY_ACCESS",
```

```
            "monitor" : true
        }
    ]
}
```

Application Elastic Load Balancing

The following example shows a component configuration in JSON format for Application Elastic Load Balancing.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "ActiveConnectionCount",
    }, {
      "alarmMetricName": "TargetResponseTime"
    }
  ],
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
        }, {
          "alarmMetricName": "StatusCheckFailed"
        }
      ],
      "logs": [
        {
          "logGroupName": "my_log_group",
          "logPath": "C:\\LogFolder\\*",
          "logType": "APPLICATION",
        }
      ],
      "windowsEvents": [
        {
          "logGroupName": "my_log_group_2",
          "eventName": "Application",
          "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
        }
      ]
    }, {
```

```
      "subComponentType": "AWS::EC2::Volume",
      "alarmMetrics": [
        {
          "alarmMetricName": "VolumeQueueLength",
        }, {
          "alarmMetricName": "BurstBalance"
        }
      ]
    }
  ],

  "alarms": [
    {
      "alarmName": "my_alb_alarm",
      "severity": "LOW"
    }
  ]
}
```

AWS Lambda Function

The following example shows a component configuration in JSON format for AWS Lambda Function.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "Errors",
      "monitor": true
    },
    {
      "alarmMetricName": "Throttles",
      "monitor": true
    },
    {
      "alarmMetricName": "IteratorAge",
      "monitor": true
    },
    {
      "alarmMetricName": "Duration",
      "monitor": true
    }
  ],
```

```
  "logs": [
    {
      "logType": "DEFAULT",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall rule group

The following example shows a component configuration in JSON format for AWS Network Firewall rule group.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Network Firewall rule group association

The following example shows a component configuration in JSON format for AWS Network Firewall rule group association.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "FirewallRuleGroupQueryVolume",
      "monitor": true
    }
  ]
}
```

AWS Step Functions

The following example shows a component configurations in JSON format for AWS Step Functions.

```
{
  "alarmMetrics": [
    {
```

```
      "alarmMetricName": "ExecutionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "LambdaFunctionsFailed",
      "monitor": true
    },
    {
      "alarmMetricName": "ProvisionedRefillRate",
      "monitor": true
    }
  ],
  "logs": [
    {
     "logGroupName": "/aws/states/HelloWorld-Logs",
      "logType": "STEP_FUNCTION",
      "monitor": true,
    }
  ]
}
```

Customer-grouped Amazon EC2 instances

The following example shows a component configuration in JSON format for customer-grouped Amazon EC2 instances.

```
{
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization",
          },
          {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION",
```

```
          }
        ],
        "processes": [
            {
                "processName": "my_process",
                "alarmMetrics": [
                    {
                        "alarmMetricName": "procstat cpu_usage",
                        "monitor": true
                    }, {
                        "alarmMetricName": "procstat memory_rss",
                        "monitor": true
                    }
                ]
            }
        ],
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ]
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength",
          }, {
            "alarmMetricName": "BurstBalance"
          }
        ]
      }
    ],
    "alarms": [
      {
        "alarmName": "my_alarm",
        "severity": "MEDIUM"
      }
    ]
  }
```

Elastic Load Balancing

The following example shows a component configuration in JSON format for Elastic Load Balancing.

```
{
    "alarmMetrics": [
      {
        "alarmMetricName": "EstimatedALBActiveConnectionCount"
      }, {
        "alarmMetricName": "HTTPCode_Backend_5XX"
      }
    ],
    "subComponents": [
      {
        "subComponentType": "AWS::EC2::Instance",
        "alarmMetrics": [
          {
            "alarmMetricName": "CPUUtilization"
          }, {
            "alarmMetricName": "StatusCheckFailed"
          }
        ],
        "logs": [
          {
            "logGroupName": "my_log_group",
            "logPath": "C:\\LogFolder\\*",
            "logType": "APPLICATION"
          }
        ],
        "processes": [
          {
            "processName": "my_process",
            "alarmMetrics": [
                {
                  "alarmMetricName": "procstat cpu_usage",
                  "monitor": true
                }, {
                  "alarmMetricName": "procstat memory_rss",
                  "monitor": true
                }
             ]
          }
        ],
```

```
        "windowsEvents": [
          {
            "logGroupName": "my_log_group_2",
            "eventName": "Application",
            "eventLevels": [ "ERROR", "WARNING", "CRITICAL" ],
            "monitor": true
          }
        ]
      }, {
        "subComponentType": "AWS::EC2::Volume",
        "alarmMetrics": [
          {
            "alarmMetricName": "VolumeQueueLength"
          }, {
            "alarmMetricName": "BurstBalance"
          }
        ]
      }
    ],

    "alarms": [
      {
        "alarmName": "my_elb_alarm",
        "severity": "HIGH"
      }
    ]
  }
```

Java

The following example shows a component configuration in JSON format for Java.

```
{
  "alarmMetrics": [ {
    "alarmMetricName": "java_lang_threading_threadcount",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_used",
    "monitor": true
  },
  {
    "alarmMetricName": "java_lang_memory_heapmemoryusage_committed",
    "monitor": true
```

```
  }],
  "logs": [ ],
  "JMXPrometheusExporter": {
      "hostPort": "8686",
      "prometheusPort": "9404"
  }
}
```

> **Note**
>
> Application Insights does not support configuring authentication for Prometheus JMX exporter. For information about how to set up authentication, see the Prometheus JMX exporter example configuration.

Kubernetes on Amazon EC2

The following example shows a component configuration in JSON format for Kubernetes on Amazon EC2.

```
{
    "alarmMetrics":[
       {
          "alarmMetricName":"cluster_failed_node_count",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_reserved_capacity",
          "monitor":true
       },
       {
          "alarmMetricName":"node_cpu_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_filesystem_utilization",
          "monitor":true
       },
       {
          "alarmMetricName":"node_memory_reserved_capacity",
          "monitor":true
       },
```

```
        {
           "alarmMetricName":"node_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"node_network_total_bytes",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_cpu_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_reserved_capacity",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_memory_utilization_over_pod_limit",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_network_rx_bytes",
           "monitor":true
        },
        {
           "alarmMetricName":"pod_network_tx_bytes",
           "monitor":true
        }
     ],
     "logs":[
        {
           "logGroupName":"/aws/containerinsights/kubernetes/application",
```

```
            "logType":"APPLICATION",
            "monitor":true,
            "encoding":"utf-8"
         }
      ],
      "subComponents":[
         {
            "subComponentType":"AWS::EC2::Instance",
            "alarmMetrics":[
               {
                  "alarmMetricName":"CPUUtilization",
                  "monitor":true
               },
               {
                  "alarmMetricName":"StatusCheckFailed",
                  "monitor":true
               },
               {
                  "alarmMetricName":"disk_used_percent",
                  "monitor":true
               },
               {
                  "alarmMetricName":"mem_used_percent",
                  "monitor":true
               }
            ],
            "logs":[
               {
                  "logGroupName":"APPLICATION-KubernetesClusterOnEC2-IAD",
                  "logPath":"",
                  "logType":"APPLICATION",
                  "monitor":true,
                  "encoding":"utf-8"
               }
            ],
            "processes" : [
              {
                  "processName" : "my_process",
                  "alarmMetrics" : [
                      {
                          "alarmMetricName" : "procstat cpu_usage",
                          "monitor" : true
                      }, {
                          "alarmMetricName" : "procstat memory_rss",
```

```
                        "monitor" : true
                    }
                ]
            }
        ]
       },
       {
          "subComponentType":"AWS::EC2::Volume",
          "alarmMetrics":[
             {
                "alarmMetricName":"VolumeReadBytes",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteBytes",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeReadOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeWriteOps",
                "monitor":true
             },
             {
                "alarmMetricName":"VolumeQueueLength",
                "monitor":true
             },
             {
                "alarmMetricName":"BurstBalance",
                "monitor":true
             }
          ]
       }
    ]
}
```

RDS MariaDB and RDS MySQL

The following example shows a component configuration in JSON format for RDS MariaDB and RDS MySQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "MYSQL",
      "monitor": true,
    },
    {
      "logType": "MYSQL_SLOW_QUERY",
      "monitor": false
    }
  ]
}
```

RDS Oracle

The following example shows a component configuration in JSON format for RDS Oracle.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "ORACLE_ALERT",
      "monitor": true,
    },
    {
      "logType": "ORACLE_LISTENER",
      "monitor": false
    }
  ]
}
```

RDS PostgreSQL

The following example shows a component configurations in JSON format for RDS PostgreSQL.

```
{
  "alarmMetrics": [
    {
      "alarmMetricName": "CPUUtilization",
      "monitor": true
    }
  ],
  "logs": [
    {
      "logType": "POSTGRESQL",
      "monitor": true
    }
  ]
}
```

SAP ASE on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_log_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_locked_login",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_data_cache_hit_ratio",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
    ],
    "sapAsePrometheusExporter": {
      "sapAseSid": "ASE",
      "sapAsePort": "4901",
      "sapAseSecretName": "ASE_DB_CREDS",
      "prometheusPort": "9399",
```

```
    "agreeToEnableASEMonitoring": true
}
```

SAP ASE High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP ASE High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "asedb_database_availability",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_trunc_log_on_chkpt_enabled",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_db_backup_age_in_days",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_last_transaction_log_backup_age_in_hours",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_suspected_database",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_db_space_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_mode",
```

```
          "monitor": true
        },
        {
          "alarmMetricName": "asedb_ha_replication_latency_in_minutes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_ASE_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2.log",
          "logType": "SAP_ASE_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_BACKUP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/ASE-*/install/SY2_BS.log",
          "logType": "SAP_ASE_BACKUP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_REP_SERVER_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/repservername/repservername.log",
          "logType": "SAP_ASE_REP_SERVER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_RMA_AGENT_LOGS-my-resource-group",
          "logPath": "/sybase/SY2/DM/RMA-*/instances/AgentContainer/logs/",
          "logType": "SAP_ASE_RMA_AGENT_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_ASE_FAULT_MANAGER_LOGS-my-resource-group",
          "logPath": "/opt/sap/FaultManager/dev_sybdbfm",
          "logType": "SAP_ASE_FAULT_MANAGER_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
  ],
```

```
  "sapAsePrometheusExporter": {
    "sapAseSid": "ASE",
    "sapAsePort": "4901",
    "sapAseSecretName": "ASE_DB_CREDS",
    "prometheusPort": "9399",
    "agreeToEnableASEMonitoring": true
}
```

SAP HANA on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_max_trigger_read_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_table_allocation_limit_used_percent",
          "monitor": true
        },
        {
```

```
          "alarmMetricName": "hanadb_cpu_usage_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_plan_cache_hit_ratio_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_last_data_backup_age_days",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_LOGS-my-resource-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.log",
          "logType": "SAP_HANA_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  }
}
```

SAP HANA High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP HANA High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "hanadb_server_startup_time_variations_seconds",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_5_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_level_4_alerts_count",
          "monitor": true
        },
        {
          "alarmMetricName": "hanadb_out_of_memory_events_count",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_stonith_enabled",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_HANA_TRACE-my-resourge-group",
          "logPath": "/usr/sap/HDB/HDB00/*/trace/*.trc",
          "logType": "SAP_HANA_TRACE",
          "monitor": true,
          "encoding": "utf-8"
        },
        {
          "logGroupName": "SAP_HANA_HIGH_AVAILABILITY-my-resource-group",
          "logPath": "/var/log/pacemaker/pacemaker.log",
          "logType": "SAP_HANA_HIGH_AVAILABILITY",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
```

```
  ],
  "hanaPrometheusExporter": {
    "hanaSid": "HDB",
    "hanaPort": "30013",
    "hanaSecretName": "HANA_DB_CREDS",
    "prometheusPort": "9668"
  },
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  }
}
```

SAP NetWeaver on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "CPUUtilization",
          "monitor": true
        },
        {
          "alarmMetricName": "StatusCheckFailed",
          "monitor": true
        },
        {
          "alarmMetricName": "disk_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "mem_used_percent",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
```

```
            "alarmMetricName": "sap_alerts_ResponseTimeDialog",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_DBRequestTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_LongRunners",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_AbortedJobs",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_BasisSystem",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Database",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Security",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_System",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_QueueTime",
            "monitor": true
          },
          {
            "alarmMetricName": "sap_alerts_Availability",
            "monitor": true
          },
          {
```

```
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_dispatcher_queue_max",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_max",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_now",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_locks_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-ML4",
          "logPath": "/usr/sap/ML4/*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "netWeaverPrometheusExporter": {
    "sapSid": "ML4",
    "instanceNumbers": [
      "00",
      "11"
```

```
    ],
    "prometheusPort": "9680"
  }
}
```

SAP NetWeaver High Availability on Amazon EC2

The following example shows a component configuration in JSON format for SAP NetWeaver High Availability on Amazon EC2.

```
{
  "subComponents": [
    {
      "subComponentType": "AWS::EC2::Instance",
      "alarmMetrics": [
        {
          "alarmMetricName": "ha_cluster_corosync_ring_errors",
          "monitor": true
        },
        {
          "alarmMetricName": "ha_cluster_pacemaker_fail_count",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_check_failover_config_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_HA_get_failover_config_HAActive",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_AbortedJobs",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Availability",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_BasisSystem",
          "monitor": true
        },
```

```
        {
          "alarmMetricName": "sap_alerts_DBRequestTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Database",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_FrontendResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_LongRunners",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_QueueTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTime",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialog",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_ResponseTimeDialogRFC",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Security",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_Shortdumps",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_alerts_SqlError",
          "monitor": true
        },
```

```
        {
          "alarmMetricName": "sap_alerts_System",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_enqueue_server_replication_state",
          "monitor": true
        },
        {
          "alarmMetricName": "sap_start_service_processes",
          "monitor": true
        }
      ],
      "logs": [
        {
          "logGroupName": "SAP_NETWEAVER_DEV_TRACE_LOGS-NetWeaver-PR1",
          "logPath": "/usr/sap/<SID>/D*/work/dev_w*",
          "logType": "SAP_NETWEAVER_DEV_TRACE_LOGS",
          "monitor": true,
          "encoding": "utf-8"
        }
      ]
    }
  ],
  "haClusterPrometheusExporter": {
    "prometheusPort": "9664"
  },
  "netWeaverPrometheusExporter": {
    "sapSid": "PR1",
    "instanceNumbers": [
      "11",
      "12"
    ],
    "prometheusPort": "9680"
  }
}
```

SQL Always On Availability Group

The following example shows a component configuration in JSON format for SQL Always On Availability Group.

```
{
```

```
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Buffer cache hit ratio",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Buffer Manager Page life expectancy",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics Processes blocked",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:General Statistics User Connections",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Locks Number of Deadlocks/sec",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "SQLServer:SQL Statistics Batch Requests/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica File Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log remaining for undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Log Send Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Mirrored Write Transaction/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Recovery Queue",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redo Bytes Remaining",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Redone Bytes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Total Log requiring undo",
      "monitor" : true
    }, {
      "alarmMetricName" : "SQLServer:Database Replica Transaction Delay",
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
```

```
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_ALWAYSON_AVAILABILITY_GROUP-<RESOURCE_GROUP_NAME>",
      "logPath" : "C:\\Program Files\\Microsoft SQL Server\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

SQL failover cluster instance

The following example shows a component configuration in JSON format for SQL failover cluster instance.

```
{
  "subComponents" : [ {
    "subComponentType" : "AWS::EC2::Instance",
    "alarmMetrics" : [ {
      "alarmMetricName" : "CPUUtilization",
      "monitor" : true
    }, {
      "alarmMetricName" : "StatusCheckFailed",
      "monitor" : true
    }, {
      "alarmMetricName" : "Processor % Processor Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory % Committed Bytes In Use",
      "monitor" : true
    }, {
      "alarmMetricName" : "Memory Available Mbytes",
      "monitor" : true
    }, {
      "alarmMetricName" : "Paging File % Usage",
      "monitor" : true
    }, {
      "alarmMetricName" : "System Processor Queue Length",
      "monitor" : true
    }, {
      "alarmMetricName" : "Network Interface Bytes Total/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "PhysicalDisk % Disk Time",
      "monitor" : true
    }, {
      "alarmMetricName" : "Bytes Received/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Normal Messages Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Urgent Message Queue Length/se",
      "monitor" : true
```

```
    }, {
      "alarmMetricName" : "Reconnect Count",
      "monitor" : true
    }, {
      "alarmMetricName" : "Unacknowledged Message Queue Length/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Outstanding",
      "monitor" : true
    }, {
      "alarmMetricName" : "Messages Sent/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Database Update Messages/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Update Messages/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Flushes/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Crypto Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Restored/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Registry Checkpoints Saved/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster API Calls/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource API Calls/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Cluster Handles/sec",
      "monitor" : true
    }, {
      "alarmMetricName" : "Resource Handles/sec",
```

```
      "monitor" : true
    } ],
    "windowsEvents" : [ {
      "logGroupName" : "WINDOWS_EVENTS-Application-<RESOURCE_GROUP_NAME>",
      "eventName" : "Application",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL"],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-System-<RESOURCE_GROUP_NAME>",
      "eventName" : "System",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL", "INFORMATION" ],
      "monitor" : true
    }, {
      "logGroupName" : "WINDOWS_EVENTS-Security-<RESOURCE_GROUP_NAME>",
      "eventName" : "Security",
      "eventLevels" : [ "WARNING", "ERROR", "CRITICAL" ],
      "monitor" : true
    } ],
    "logs" : [ {
      "logGroupName" : "SQL_SERVER_FAILOVER_CLUSTER_INSTANCE-<RESOURCE_GROUP_NAME>",
      "logPath" : "\\\\amznfsxjmzbykwn.mydomain.aws\\SQLDB\\MSSQL**.MSSQLSERVER\\MSSQL\
\Log\\ERRORLOG",
      "logType" : "SQL_SERVER",
      "monitor" : true,
      "encoding" : "utf-8"
    } ]
  }, {
    "subComponentType" : "AWS::EC2::Volume",
    "alarmMetrics" : [ {
      "alarmMetricName" : "VolumeReadBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteBytes",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeReadOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeWriteOps",
      "monitor" : true
    }, {
    "alarmMetricName" : "VolumeQueueLength",
      "monitor" : true
    }, {
```

```
    "alarmMetricName" : "VolumeThroughputPercentage",
      "monitor" : true
    }, {
    "alarmMetricName" : "BurstBalance",
      "monitor" : true
    } ]
  } ]
}
```

Create and configure CloudWatch Application Insights monitoring using CloudFormation templates

You can add Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from AWS CloudFormation templates.

This section provides sample CloudFormation templates in both JSON and YAML formats to help you create and configure Application Insights monitoring.

To view the Application Insights resource and property reference in the *CloudFormation User Guide*, see ApplicationInsights resource type reference.

Sample templates

- Create an Application Insights application for the entire CloudFormation stack
- Create an Application Insights application with detailed settings
- Create an Application Insights application with CUSTOM mode component configuration
- Create an Application Insights application with DEFAULT mode component configuration
- Create an Application Insights application with DEFAULT_WITH_OVERWRITE mode component configuration

Create an Application Insights application for the entire CloudFormation stack

To apply the following template, you must create AWS resources and one or more resource groups from which to create Application Insights applications to monitor those resources. For more information, see Getting started with AWS Resource Groups.

The first two parts of the following template specify a resource and a resource group. The last part of the template creates an Application Insights application for the resource group, but does not

configure the application or apply monitoring. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Resource Group stack",
    "Resources": {
        "EC2Instance": {
            "Type": "AWS::EC2::Instance",
            "Properties": {
                "ImageId" : "ami-abcd1234efgh5678i",
                "SecurityGroupIds" : ["sg-abcd1234"]
            }
        },
        ...
        "ResourceGroup": {
            "Type": "AWS::ResourceGroups::Group",
            "Properties": {
                "Name": "my_resource_group"
            }
        },
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group"
            },
            "DependsOn" : "ResourceGroup"
        }
    }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Resource Group stack
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-abcd1234efgh5678i
```

```
      SecurityGroupIds:
      - sg-abcd1234
  ...
  ResourceGroup:
    Type: AWS::ResourceGroups::Group
    Properties:
      Name: my_resource_group
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
    DependsOn: ResourceGroup
```

The following template section applies the default monitoring configuration to the Application Insights application. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.

When `AutoConfigurationEnabled` is set to `true`, all components of the application are configured with the recommended monitoring settings for the `DEFAULT` application tier. For more information about these settings and tiers, see DescribeComponentConfigurationRecommendation and UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Test Application Insights Application stack",
    "Resources": {
        "AppInsightsApp": {
            "Type": "AWS::ApplicationInsights::Application",
            "Properties": {
                "ResourceGroupName": "my_resource_group",
                "AutoConfigurationEnabled": true
            }
        }
    }
}
```

Template in YAML format

```
---
AWSTemplateFormatVersion: '2010-09-09'
```

```
Description: Test Application Insights Application stack
Resources:
  AppInsightsApp:
    Type: AWS::ApplicationInsights::Application
    Properties:
      ResourceGroupName: my_resource_group
      AutoConfigurationEnabled: true
```

Create an Application Insights application with detailed settings

The following template performs these actions:

- Creates an Application Insights application with CloudWatch Events notification and OpsCenter enabled. For more information, see the CreateApplication command details in the *Amazon CloudWatch Application Insights API Reference*.
- Tags the application with two tags, one of which has no tag values. For more information, see TagResource in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two custom instance group components. For more information, see CreateComponent in the *Amazon CloudWatch Application Insights API Reference*.
- Creates two log pattern sets. For more information, see CreateLogPattern in the *Amazon CloudWatch Application Insights API Reference*.
- Sets `AutoConfigurationEnabled` to `true`, which configures all components of the application with the recommended monitoring settings for the `DEFAULT` tier. For more information, see DescribeComponentConfigurationRecommendation in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "CWEMonitorEnabled": true,
        "OpsCenterEnabled": true,
        "OpsItemSNSTopicArn": "arn:aws:sns:us-east-1:123456789012:my_topic",
        "AutoConfigurationEnabled": true,
        "Tags": [
            {
                "Key": "key1",
```

```
                "Value": "value1"
            },
            {
                "Key": "key2",
                "Value": ""
            }
        ],
        "CustomComponents": [
            {
                "ComponentName": "test_component_1",
                "ResourceList": [
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            },
            {
                "ComponentName": "test_component_2",
                "ResourceList": [
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i",
                    "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i"
                ]
            }
        ],
        "LogPatternSets": [
            {
                "PatternSetName": "pattern_set_1",
                "LogPatterns": [
                    {
                        "PatternName": "deadlock_pattern",
                        "Pattern": ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))",
                        "Rank": 1
                    }
                ]
            },
            {
                "PatternSetName": "pattern_set_2",
                "LogPatterns": [
                    {
                        "PatternName": "error_pattern",
                        "Pattern": ".*[\\s\\[]ERROR[\\s\\]].*",
                        "Rank": 1
                    },
                    {
                        "PatternName": "warning_pattern",
                        "Pattern": ".*[\\s\\[]WARN(ING)?[\\s\\]].*",
```

```
                        "Rank": 10
                    }
                ]
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  CWEMonitorEnabled: true
  OpsCenterEnabled: true
  OpsItemSNSTopicArn: arn:aws:sns:us-east-1:123456789012:my_topic
  AutoConfigurationEnabled: true
  Tags:
  - Key: key1
    Value: value1
  - Key: key2
    Value: ''
  CustomComponents:
  - ComponentName: test_component_1
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  - ComponentName: test_component_2
    ResourceList:
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
    - arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1234efgh5678i
  LogPatternSets:
  - PatternSetName: pattern_set_1
    LogPatterns:
    - PatternName: deadlock_pattern
      Pattern: ".*\\sDeadlocked\\sSchedulers(([^\\w].*)|($))"
      Rank: 1
  - PatternSetName: pattern_set_2
    LogPatterns:
    - PatternName: error_pattern
      Pattern: ".*[\\s\\[]ERROR[\\s\\]].*"
      Rank: 1
    - PatternName: warning_pattern
```

```
      Pattern: ".*[\\s\\[]WARN(ING)?[\\s\\]].*"
      Rank: 10
```

Create an Application Insights application with CUSTOM mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see CreateApplication in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `CUSTOM`, which causes this component to be configured with the configuration specified in `CustomComponentConfiguration`. For more information, see UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group,
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "CUSTOM",
                "CustomComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            },
                            ...
                        ],
                        "Logs": [
                            {
                                "LogGroupName": "my_log_group_1",
                                "LogPath": "C:\\LogFolder_1\\*",
                                "LogType": "DOT_NET_CORE",
                                "Encoding": "utf-8",
                                "PatternSet": "my_pattern_set_1"
```

```
                        },
                        ...
                    ],
                    "WindowsEvents": [
                        {
                            "LogGroupName": "my_windows_event_log_group_1",
                            "EventName": "Application",
                            "EventLevels": [
                                "ERROR",
                                "WARNING",
                                ...
                            ],
                            "Encoding": "utf-8",
                            "PatternSet": "my_pattern_set_2"
                        },
                        ...
                    ],
                    "Alarms": [
                        {
                            "AlarmName": "my_alarm_name",
                            "Severity": "HIGH"
                        },
                        ...
                    ]
                },
                "SubComponentTypeConfigurations": [
                    {
                        "SubComponentType": "EC2_INSTANCE",
                        "SubComponentConfigurationDetails": {
                            "AlarmMetrics": [
                                {
                                    "AlarmMetricName": "DiskReadOps"
                                },
                                ...
                            ],
                            "Logs": [
                                {
                                    "LogGroupName": "my_log_group_2",
                                    "LogPath": "C:\\LogFolder_2\\*",
                                    "LogType": "IIS",
                                    "Encoding": "utf-8",
                                    "PatternSet": "my_pattern_set_3"
                                },
                                ...
```

```
                                ],
                                "processes" : [
                                    {
                                        "processName" : "my_process",
                                        "alarmMetrics" : [
                                    {
                                        "alarmMetricName" : "procstat cpu_usage",
                                        "monitor" : true
                                    }, {
                                        "alarmMetricName" : "procstat memory_rss",
                                        "monitor" : true
                                    }
                                ]
                            }
                        ],
                                "WindowsEvents": [
                                    {
                                        "LogGroupName": "my_windows_event_log_group_2",
                                        "EventName": "Application",
                                        "EventLevels": [
                                            "ERROR",
                                            "WARNING",
                                            ...
                                        ],
                                        "Encoding": "utf-8",
                                        "PatternSet": "my_pattern_set_4"
                                    },
                                    ...
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
```

```
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: CUSTOM
    CustomComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
        ...
        Logs:
        - LogGroupName: my_log_group_1
          LogPath: C:\LogFolder_1\*
          LogType: DOT_NET_CORE
          Encoding: utf-8
          PatternSet: my_pattern_set_1
        ...
        WindowsEvents:
        - LogGroupName: my_windows_event_log_group_1
          EventName: Application
          EventLevels:
          - ERROR
          - WARNING
          ...
          Encoding: utf-8
          PatternSet: my_pattern_set_2
        ...
        Alarms:
        - AlarmName: my_alarm_name
          Severity: HIGH
        ...
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          AlarmMetrics:
          - AlarmMetricName: DiskReadOps
          ...
          Logs:
          - LogGroupName: my_log_group_2
            LogPath: C:\LogFolder_2\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set_3
          ...
```

```
          Processes:
          - ProcessName: my_process
            AlarmMetrics:
            - AlarmMetricName: procstat cpu_usage
              ...
            ...
          WindowsEvents:
          - LogGroupName: my_windows_event_log_group_2
            EventName: Application
            EventLevels:
            - ERROR
            - WARNING
            ...
            Encoding: utf-8
            PatternSet: my_pattern_set_4
          ...
```

Create an Application Insights application with `DEFAULT` mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see CreateApplication in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT` and `Tier` to `SQL_SERVER`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `SQL_Server` tier. For more information, see DescribeComponentConfiguration and UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentARN": "my_component",
                "Tier": "SQL_SERVER",
                "ComponentConfigurationMode": "DEFAULT"
```

```
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
  ComponentMonitoringSettings:
  - ComponentARN: my_component
    Tier: SQL_SERVER
    ComponentConfigurationMode: DEFAULT
```

Create an Application Insights application with `DEFAULT_WITH_OVERWRITE` mode component configuration

The following template performs these actions:

- Creates an Application Insights application. For more information, see CreateApplication in the *Amazon CloudWatch Application Insights API Reference*.
- Component `my_component` sets `ComponentConfigurationMode` to `DEFAULT_WITH_OVERWRITE` and `tier` to `DOT_NET_CORE`, which causes this component to be configured with the configuration settings that Application Insights recommends for the `DOT_NET_CORE` tier. Overwritten configuration settings are specified in the `DefaultOverwriteComponentConfiguration`:
  - At the component level `AlarmMetrics` settings are overwritten.
  - At the sub-component level, for the `EC2_Instance` type sub-components, `Logs` settings are overwritten.

  For more information, see UpdateComponentConfiguration in the *Amazon CloudWatch Application Insights API Reference*.

Template in JSON format

```
{
    "Type": "AWS::ApplicationInsights::Application",
```

```
    "Properties": {
        "ResourceGroupName": "my_resource_group",
        "ComponentMonitoringSettings": [
            {
                "ComponentName": "my_component",
                "Tier": "DOT_NET_CORE",
                "ComponentConfigurationMode": "DEFAULT_WITH_OVERWRITE",
                "DefaultOverwriteComponentConfiguration": {
                    "ConfigurationDetails": {
                        "AlarmMetrics": [
                            {
                                "AlarmMetricName": "StatusCheckFailed"
                            }
                        ]
                    },
                    "SubComponentTypeConfigurations": [
                        {
                            "SubComponentType": "EC2_INSTANCE",
                            "SubComponentConfigurationDetails": {
                                "Logs": [
                                    {
                                        "LogGroupName": "my_log_group",
                                        "LogPath": "C:\\LogFolder\\*",
                                        "LogType": "IIS",
                                        "Encoding": "utf-8",
                                        "PatternSet": "my_pattern_set"
                                    }
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

Template in YAML format

```
---
Type: AWS::ApplicationInsights::Application
Properties:
  ResourceGroupName: my_resource_group
```

```
  ComponentMonitoringSettings:
  - ComponentName: my_component
    Tier: DOT_NET_CORE
    ComponentConfigurationMode: DEFAULT_WITH_OVERWRITE
    DefaultOverwriteComponentConfiguration:
      ConfigurationDetails:
        AlarmMetrics:
        - AlarmMetricName: StatusCheckFailed
      SubComponentTypeConfigurations:
      - SubComponentType: EC2_INSTANCE
        SubComponentConfigurationDetails:
          Logs:
          - LogGroupName: my_log_group
            LogPath: C:\LogFolder\*
            LogType: IIS
            Encoding: utf-8
            PatternSet: my_pattern_set
```

Tutorial: Set up monitoring for SAP ASE

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP ASE databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and help mean time to resolution (MTTR) for your SAP ASE databases.

Application Insights for SAP ASE topics

- Supported environments
- Supported operating systems
- Features
- Prerequisites
- Set up monitoring on your SAP ASE database
- Manage monitoring of your SAP ASE database
- Configure the alarm threshold
- View and troubleshoot SAP ASE problems detected by Application Insights
- Troubleshooting Application Insights for SAP ASE

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns. You provide and install SAP ASE database software and supported SAP application software.

- **One or more SAP ASE databases on a single Amazon EC2 instance** – SAP ASE in a single-node, scale-up architecture.
- **Cross-AZ SAP ASE database high availability setup** – SAP ASE with high availability configured across two Availability Zones using SUSE/RHEL clustering.

> **Note**
>
> CloudWatch Application Insights supports only single SAP system ID (SID) ASE HA environments. If multiple ASE HA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP ASE supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4
- SuSE Linux 12 SP5
- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 SP3
- SuSE Linux 15 SP4
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP
- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- RedHat Linux 7.6
- RedHat Linux 7.7
- RedHat Linux 7.9
- RedHat Linux 8.1
- RedHat Linux 8.4
- RedHat Linux 8.6

Features

CloudWatch Application Insights for SAP ASE provides the following features:

- Automatic SAP ASE workload detection
- Automatic SAP ASE alarm creation based on static threshold
- Automatic SAP ASE alarm creation based on anomaly detection
- Automatic SAP ASE log pattern recognition
- Health dashboard for SAP ASE
- Problem dashboard for SAP ASE

Prerequisites

You must perform the following prerequisites to configure an SAP ASE database with CloudWatch Application Insights:

- **SAP ASE configuration parameters** – The following configuration parameters must be enabled on your ASE DB: `"enable monitoring"`, `"sql text pipe max messages"`, `"sql text pipe active"`. This allows CloudWatch Application Insights to provide full monitoring capabilities for your DB. If these settings aren't enabled on your ASE database, Application Insights will automatically enable them to collect the necessary metrics to allow monitoring.
- **SAP ASE database user** – The database user provided during Application Insights onboarding must have permission to access the following:
  - System tables in the master database and user (tenant) databases
  - Monitoring tables

- **SAPHostCtrl** – Install and set up SAPHostCtrl on your Amazon EC2 instance.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see Manually create or edit the CloudWatch agent configuration file.
- **AWS Systems Manager enablement** – Install SSM Agent on your instances, and enable the instances enabled for SSM. For information about how to install the SSM agent, see Working with SSM Agent in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your database.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see AWS Systems Manager identity-based policy examples.
  - You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with Amazon CloudWatch agent.
  - You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in AWS Secrets Manager. For more information about inline policies, see Inline policies in the *AWS Identity and Access Management User Guide*.

    JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource":
 "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- **AWS Resource Groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch

Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP ASE database. If there are multiple databases per account, we recommend that you create one resource group that includes the AWS resources for each SAP ASE database system.

- **IAM permissions** – For non-admin users:
  - You must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see IAM policy for CloudWatch Application Insights.
  - The user must have permission to create a secret in AWS Secrets Manager to store the database user credentials. For more information, see Example: Permission to create secrets.

    JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.

Set up monitoring on your SAP ASE database

Use the following steps to set up monitoring for your SAP ASE database

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.

4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group that contains your SAP ASE database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the *AWS Resource Groups User Guide.*
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
6. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *AWS Resource Groups and Tags User Guide*.
8. Choose **Next** to continue to set up monitoring.
9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

   > **Note**
   >
   > Components that contain a detected SAP ASE High Availability workload support only one workload on a component. Components that contain a detected SAP ASE single node workload support multiple workloads, but you can't add or remove workloads. All automatically detected workloads will be monitored.

10. Choose **Next**.
11. On the **Specify component details** page, enter the username and password of your SAP ASE databases.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**.

If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP ASE system, which can take up to 2 hours.

Manage monitoring of your SAP ASE database

You can manage user credentials, metrics, and log paths for your SAP ASE database by performing the following steps:

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
6. Under **Metrics**, you can choose the SAP ASE metrics according to your requirements. SAP ASE metric names are prefixed with `asedb`. You can add up to 60 metrics per component.
7. Under **ASE configuration**, enter the username and password for the SAP ASE database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP ASE database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Configure the alarm threshold

CloudWatch Application Insights automatically creates a Amazon CloudWatch metric for the alarm to watch, along with the threshold for that metric. The alarm changes to the `ALARM` state when the metric surpasses the threshold for a specified number of evaluation periods. Note that these settings are not retained by Application Insights.

To edit an alarm for a single metric, perform the following steps:

1. Open the CloudWatch console.
2. In the left navigation pane, choose **Alarms**>**All alarms**.
3. Select the radio button next to the alarm that was automatically created by CloudWatch Application Insights. Then choose **Actions**, and select **Edit** from the dropdown menu.
4. Edit the following parameters under **Metric**.
   a. Under **Statistic**, choose one of the statistics or predefined percentiles, or specify a custom percentile. For example, `p95.45`.
   b. Under **Period**, choose the evaluations period for the alarm. When you evaluate the alarm, each period is aggregated into one data point.
5. Edit the following parameters under **Conditions**.
   a. Choose whether the metric must be greater than, less than, or equal to the threshold.
   b. Specify the threshold value.
6. Under **Additional configuration** edit the following parameters.
   a. Under **Datapoints to alarm**, specify the number of data points, or evaluation periods, that must be in the `ALARM` state to initiate the alarm. When the two values match, an alarm is created that enters `ALARM` state if the designated number of consecutive periods are exceeded. To create an `m` out of `n` alarm, specify a lower value for the first data point than for the second. For more information about evaluating alarms, see Evaluating an alarm.
   b. Under **Missing data treatment**, choose the behavior of the alarm when some data points are missing. For more information about missing data treatment, see Configuring how CloudWatch alarms treat missing data.
   c. If the alarm uses a percentile as the monitored statistic, a **Percentiles with low samples** box appears. Choose whether to evaluate or ignore cases with low sample rates. If you choose **ignore (maintain alarm state)**, the current alarm state is always maintained when the sample size is too low. For more information about percentiles with low samples, see Percentile-based alarms and low data samples.
7. Choose **Next**.
8. Under **Notification**, select an SNS topic to notify when the alarm is in `ALARM` state, `OK` state, or `INSUFFICIENT_DATA` state.
9. Choose **Update alarm**.

View and troubleshoot SAP ASE problems detected by Application Insights

This section helps you resolve common troubleshooting problems that occur when you configure monitoring for SAP ASE on Application Insights.

SAP ASE Backup Server errors

You can identify the error message by checking the dynamically created dashboard. The dashboard shows the error message reported in the SAP ASE Backup Server. For more details about SAP ASE Backup Server logs, see SAP Documentation Backup Server Error Logging.

SAP ASE long running transactions

Identify the long running transaction and confirm whether it can be stopped or if the running time is intentional. For more details, see 2180410 — How to display transaction log records for long running transactions? — SAP ASE.

SAP ASE User connections

Review whether your SAP ASE database is sized accordingly for the workload you intend to run on the database. For more details, see Configuring User Connections in the SAP documentation.

SAP ASE disk space

You can identify the database layer that is causing the problem by checking the dynamically created dashboard. The dashboard shows the related metrics and log file snippets. It is important to understand the cause of the disk growth and when applicable, increase the physical disk size, the allocated disk space, or both. For more details, see SAP Documentation disk resize in the SAP documentation.

Troubleshooting Application Insights for SAP ASE

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Error	Error returned	Root cause	Resolution
Unable to add more than 60 monitor metrics.	`Component cannot have more than`	The current metric limit is 60	Remove unnecessary metrics to adhere to the limit.

Error	Error returned	Root cause	Resolution
	`60 monitored metric`	monitored metrics per component.	
No SAP metrics or alarms appear after the onboarding process	The `run` command on the `AWS-ConfigureAWSPackage` failed in AWS Systems Manager. The output shows the error: `CT-LIBRARY error:ct_connect(): protocol specific layer: external error: The attempt to connect to the server failed`	The username and password might be incorrect.	Verify that the username and password are valid, then rerun the onboarding process.

Tutorial: Set up monitoring for SAP HANA

This tutorial demonstrates how to configure CloudWatch Application Insights to set up monitoring for your SAP HANA databases. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and facilitate mean time to resolution (MTTR) for your SAP HANA databases.

Application Insights for SAP HANA topics

- Supported environments
- Supported operating systems
- Features
- Prerequisites
- Set up your SAP HANA database for monitoring
- Manage monitoring of your SAP HANA database
- View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights

- Anomaly detection for SAP HANA
- Troubleshooting Application Insights for SAP HANA

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns. You provide and install SAP HANA database software and supported SAP application software.

- **SAP HANA database on a single Amazon EC2 instance** — SAP HANA in a single-node, scale-up architecture, with up to 24TB of memory.
- **SAP HANA database on multiple Amazon EC2 instances** — SAP HANA in a multi-node, scale-out architecture.
- **Cross-AZ SAP HANA database high availability setup** — SAP HANA with high availability configured across two Availability Zones using SUSE/RHEL clustering.

> **Note**
>
> CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

Supported operating systems

CloudWatch Application Insights for SAP HANA supports x86-64 architecture on the following operating systems:

- SuSE Linux 12 SP4 For SAP
- SuSE Linux 12 SP5 For SAP
- SuSE Linux 15
- SuSE Linux 15 SP1
- SuSE Linux 15 SP2
- SuSE Linux 15 For SAP
- SuSE Linux 15 SP1 For SAP
- SuSE Linux 15 SP2 For SAP

- SuSE Linux 15 SP3 For SAP
- SuSE Linux 15 SP4 For SAP
- SuSE Linux 15 SP5 For SAP
- RedHat Linux 8.6 For SAP With High Availability and Update Services
- RedHat Linux 8.5 For SAP With High Availability and Update Services
- RedHat Linux 8.4 For SAP With High Availability and Update Services
- RedHat Linux 8.3 For SAP With High Availability and Update Services
- RedHat Linux 8.2 For SAP With High Availability and Update Services
- RedHat Linux 8.1 For SAP With High Availability and Update Services
- RedHat Linux 7.9 For SAP With High Availability and Update Services

Features

CloudWatch Application Insights for SAP HANA provides the following features:

- Automatic SAP HANA workload detection
- Automatic SAP HANA alarm creation based on static threshold
- Automatic SAP HANA alarm creation based on anomaly detection
- Automatic SAP HANA log pattern recognition
- Health dashboard for SAP HANA
- Problem dashboard for SAP HANA

Prerequisites

You must perform the following prerequisites to configure an SAP HANA database with CloudWatch Application Insights:

- **SAP HANA** – Install a running and reachable SAP HANA database 2.0 SPS05 on an Amazon EC2 instance.
- **SAP HANA database user** – A database user with monitoring roles must be created in the SYSTEM database and all tenants.

  Example

The following SQL commands create a user with monitoring roles.

```
su - <sid>adm
hdbsql -u SYSTEM -p <SYSTEMDB password> -d SYSTEMDB
CREATE USER CW_HANADB_EXPORTER_USER PASSWORD <Monitoring user password> NO
 FORCE_FIRST_PASSWORD_CHANGE;
CREATE ROLE CW_HANADB_EXPORTER_ROLE;
GRANT MONITORING TO CW_HANADB_EXPORTER_ROLE;
GRANT CW_HANADB_EXPORTER_ROLE TO CW_HANADB_EXPORTER_USER;
```

- **Python 3.8** – Install Python 3.8 or later versions on your operating system. Use the latest release of Python. If Python3 is not detected on your operating system, Python 3.6 will be installed.

  For more information, see the installation example.

  > **Note**
  >
  > Manual installation of Python 3.8 or higher is required for SuSE Linux 15 SP4, RedHat Linux 8.6, and later operating systems.

- **Pip3** – Install the installer program, pip3, on your operating system. If pip3 is not detected on your operating system, it will be installed.
- **hdbclient** – CloudWatch Application Insights uses the python driver to connect to the SAP HANA database. If the client is not installed under python3, make sure that you have hdbclient tar file version `2.10 or later` under `/hana/shared/SID/hdbclient/`.
- **Amazon CloudWatch agent** – Make sure that you are not running a preexisting CloudWatch agent on your Amazon EC2 instance. If you have CloudWatch agent installed, make sure to remove the configuration of the resources you are using in CloudWatch Application Insights from the existing CloudWatch agent configuration file to avoid a merge conflict. For more information, see Manually create or edit the CloudWatch agent configuration file.
- **AWS Systems Manager enablement** – Install SSM Agent on your instances, and the instances must be enabled for SSM. For information about how to install the SSM Agent, see Working with SSM Agent in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your database.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see AWS Systems Manager identity-based policy examples.

- You must attach the `CloudWatchAgentServerPolicy` to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with CloudWatch agent.
- You must attach the following IAM inline policy to the Amazon EC2 instance role to read the password stored in AWS Secrets Manager. For more information about inline policies, see Inline policies in the *AWS Identity and Access Management User Guide*.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource":
 "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
        }
    ]
}
```

- **AWS resource groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances and Amazon EBS volumes running your SAP HANA database. If there are multiple databases per account, we recommend that you create one resource group that includes the AWS resources for each SAP HANA database system.
- **IAM permissions** – For non-admin users:
  - You must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role, and attach it to your user identity. For steps to attach the policy, see IAM policy for CloudWatch Application Insights.
  - The user must have permission to create a secret in AWS Secrets Manager to store the database user credentials. For more information, see Example: Permission to create secrets.

    JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret"
      ],
      "Resource": "arn:aws:secretsmanager:*:*:secret:ApplicationInsights-*"
    }
  ]
}
```

- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.

Set up your SAP HANA database for monitoring

Use the following steps to set up monitoring for your SAP HANA database

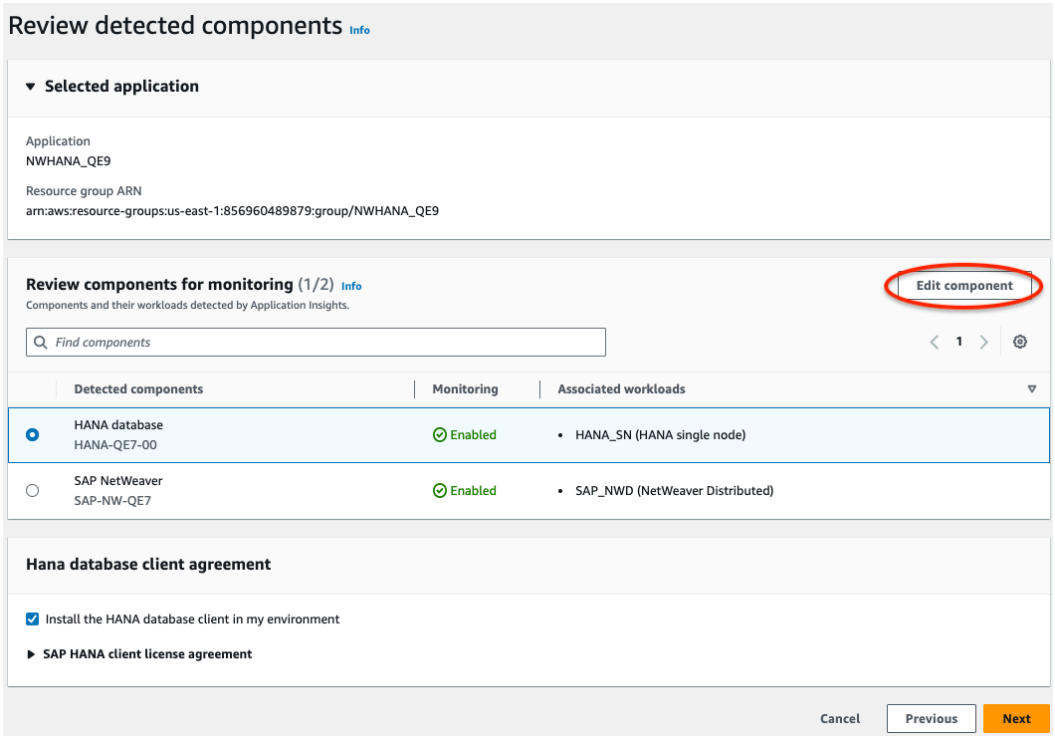
1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, choose **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group that contains your SAP HANA database resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown. For more information about creating resource groups, see the *AWS Resource Groups User Guide.*
5. Under **Monitor CloudWatch Events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
6. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations

that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.

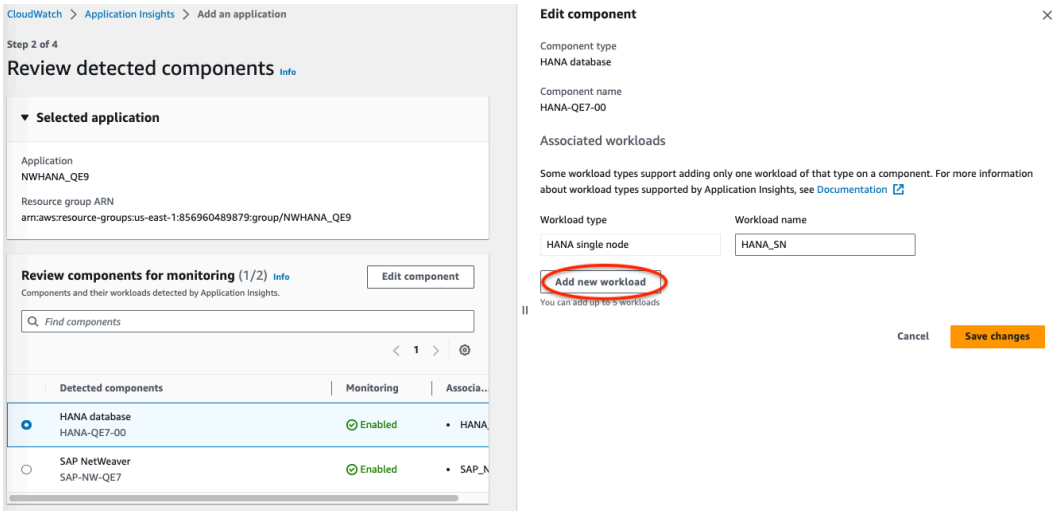
7. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *AWS Resource Groups and Tags User Guide*.
8. Choose **Next** to continue to set up monitoring.
9. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

    a. To add workloads to a component that contains a detected SAP HANA single node workload, select the component, then choose **Edit component**.

> **Note**
>
> Components that contain a detected SAP HANA multi node or HANA High Availability workload support only one workload on a component.



    b. To add a new workload, choose **Add new workload**.



c. When you are finished editing workloads, choose **Save changes**.

10. Choose **Next**.
11. On the **Specify component details** page, enter the username and password.
12. Review your application monitoring configuration, and choose **Submit**.
13. The application details page opens, where you can view the **Application summary**, the list of **Monitored components and workloads**, and **Unmonitored components and workloads**. If you select the radio button next to a component or workload, you can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. When you submit your configuration, your account deploys all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

Manage monitoring of your SAP HANA database

You can manage user credentials, metrics, and log paths for your SAP HANA database by performing the following steps:

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, choose **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
4. Under **Monitored components**, select the radio button next to the component name. Then, choose **Manage monitoring**.
5. Under **EC2 instance group logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.

6. Under **Metrics**, you can choose the SAP HANA metrics according to your requirements. SAP HANA metric names are prefixed with hanadb. You can add up to 40 metrics per component.
7. Under **HANA configuration**, enter the password and user name for the SAP HANA database. This is the username and password that Amazon CloudWatch agent uses to connect to the SAP HANA database.
8. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
9. Review your application monitoring configuration and choose **Submit**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP HANA system, which can take up to 2 hours.

View and troubleshoot SAP HANA problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP HANA on Application Insights.

Troubleshooting topics

- SAP HANA database reaches memory allocation limit
- Disk full event
- SAP HANA backup stopped running

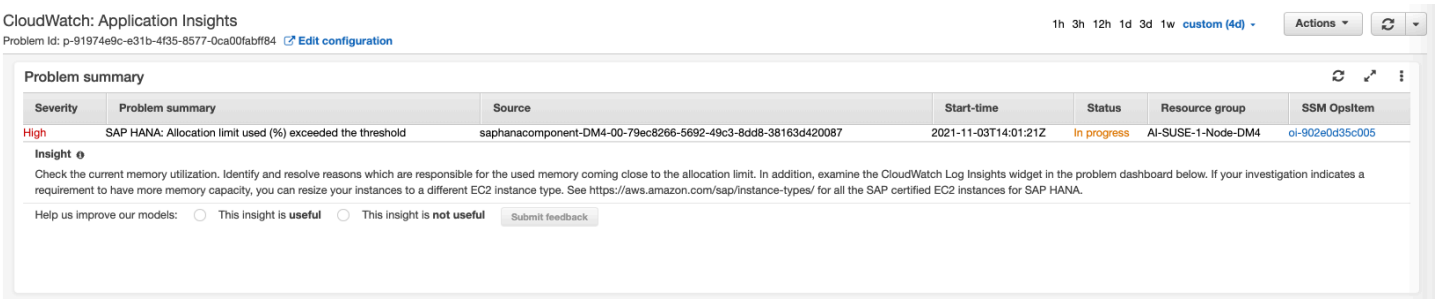
SAP HANA database reaches memory allocation limit

Description

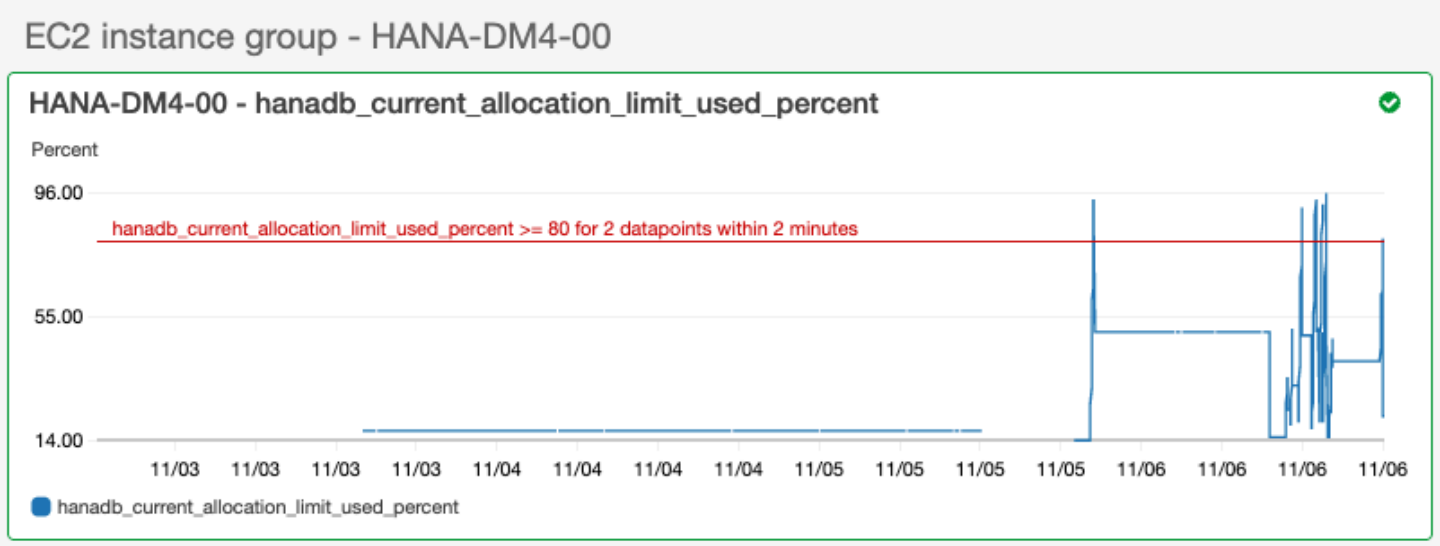
Your SAP application that is backed by an SAP HANA database malfunctions because of high memory pressure, leading to application performance degradation.

Resolution

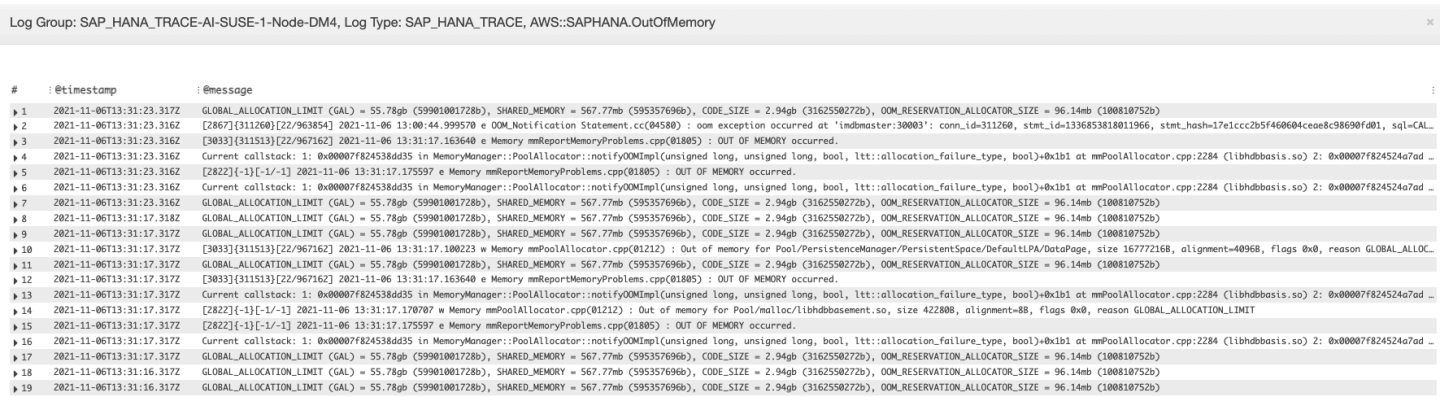
You can identify the application layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be because of a large data load in the SAP HANA system.



The used memory allocation exceeds the threshold of 80 percent of the total memory allocation limit.



The log group shows the scheme `BNR-DATA` and table `IMDBMASTER_30003` ran out of memory. In addition, the log group shows the exact time of the issue, current global location limit, shared memory, code size, and OOM reservation allocation size.



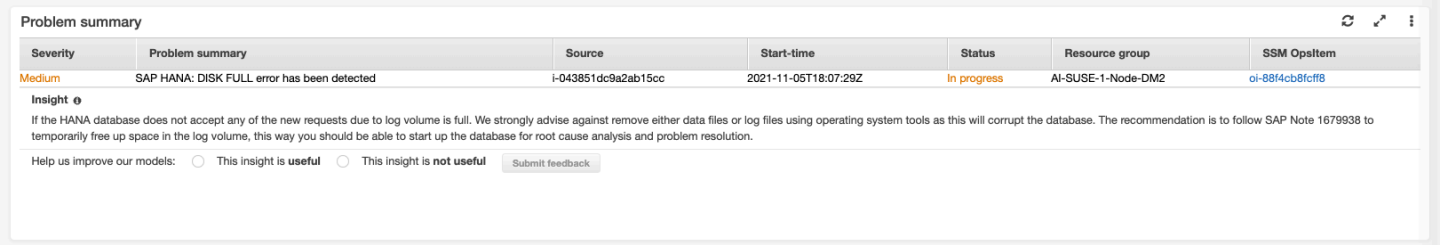
Disk full event

Description

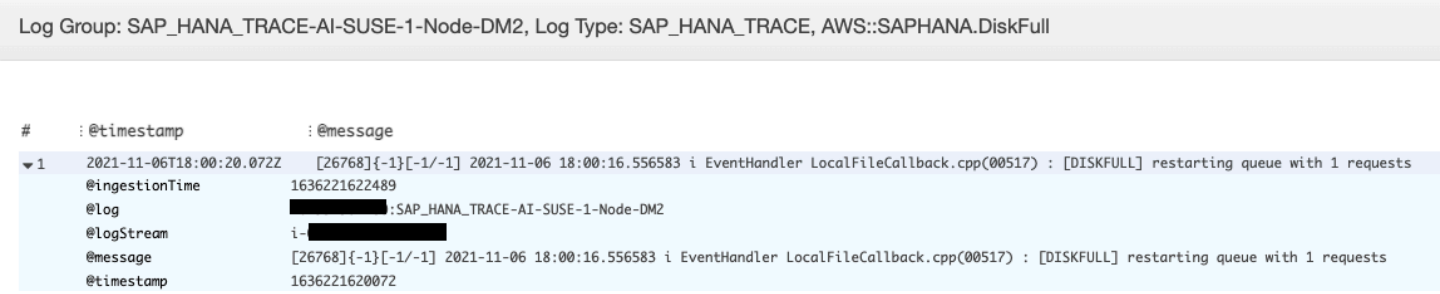
Your SAP application that is backed by an SAP HANA database stops responding, which leads to an inability to access the database.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets. In the following example, the problem may be that the administrator failed to enable automatic log backup, which caused the sap/hana/log directory to fill up.



The log group widget in the problem dashboard shows the `DISKFULL` event.



SAP HANA backup stopped running

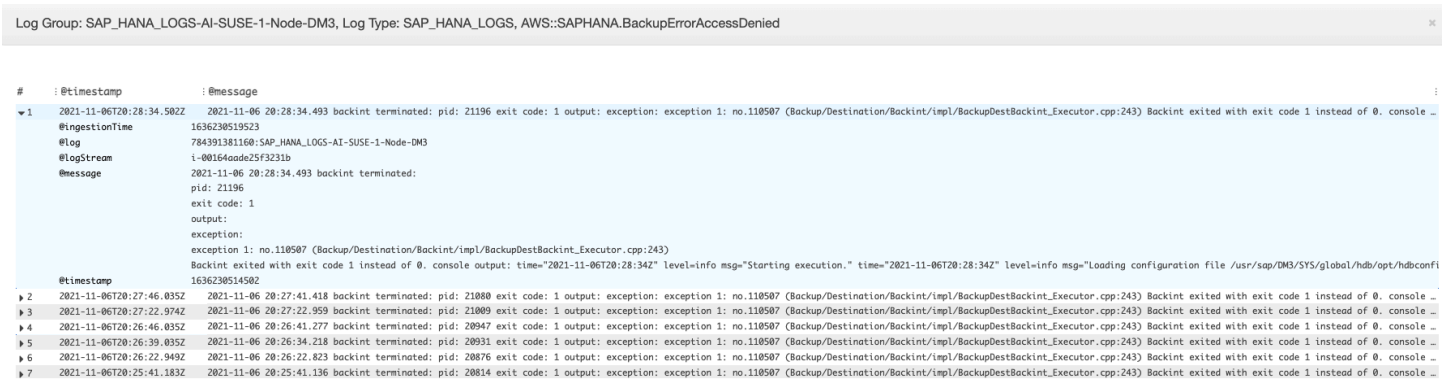
Description

Your SAP application that is backed by an SAP HANA database has stopped working.

Resolution

You can identify the database layer that is causing the problem by checking the dynamically created dashboard, which shows the related metrics and log file snippets.

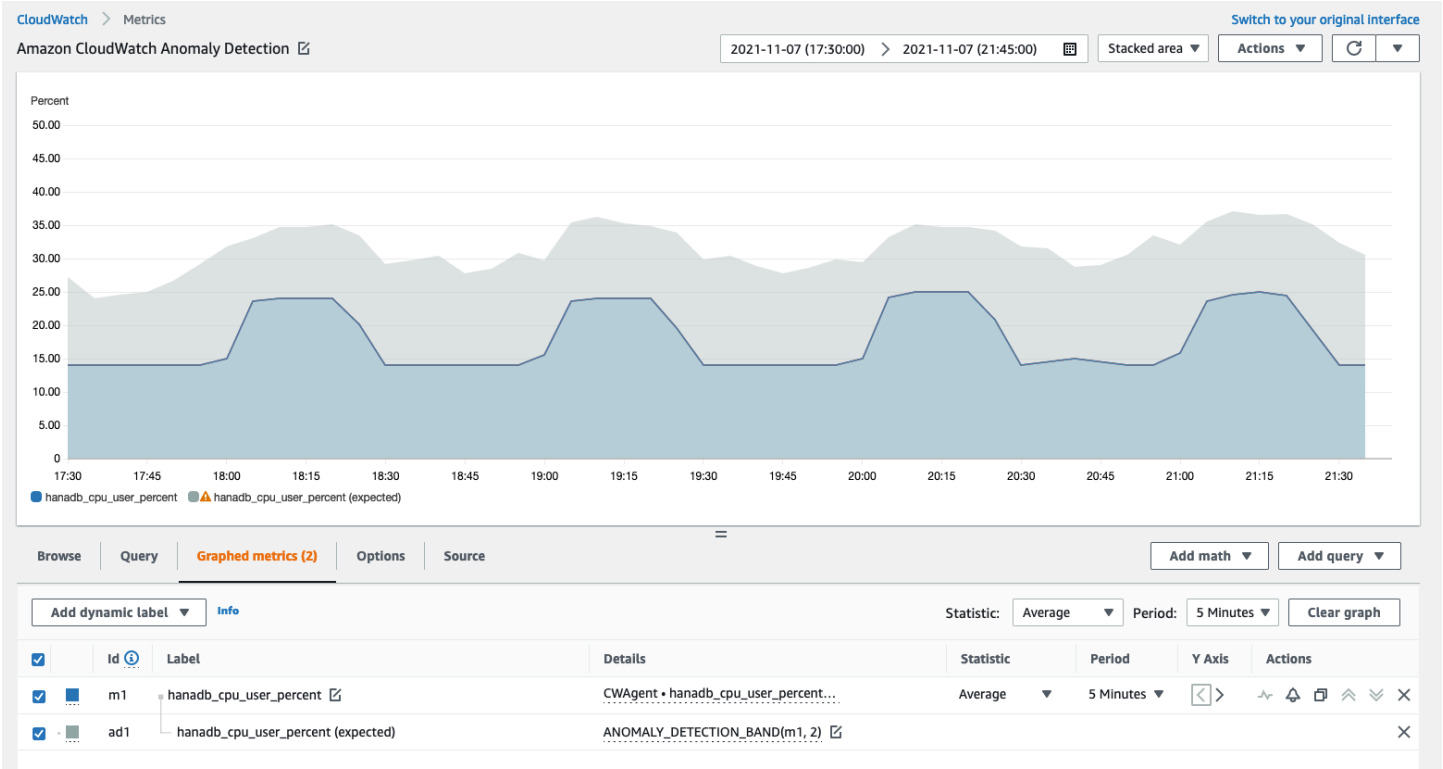
The log group widget in the problem dashboard shows the `ACCESS DENIED` event. This includes additional information, such as the S3 bucket, the S3 bucket folder, and the S3 bucket Region.



Anomaly detection for SAP HANA

For specific SAP HANA metrics, such as the number of thread count, CloudWatch applies statistical and machine learning algorithms to define the threshold. These algorithms continuously analyze the metrics of the SAP HANA database, determine normal baselines, and surface anomalies with minimal user intervention. The algorithms generate an anomaly detection model, which generates a range of expected values that represent normal metric behavior.

Anomaly detection algorithms account for the seasonality and trend changes of metrics. The seasonality changes can be hourly, daily, or weekly, as shown in the following examples of the SAP HANA CPU usage.



After you create a model, CloudWatch anomaly detection continuously evaluates the model and makes adjustments to it to make sure that is it as accurate as possible. This includes retraining the model to adjust if the metric values evolve over time or experience sudden changes. It also includes predictors to improve the models for metrics that are seasonal, spiky, or sparse.

Troubleshooting Application Insights for SAP HANA

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitored metrics

The output shows the following error.

```
Component cannot have more than 60 monitored metrics
```

Root cause – The current metric limit is 60 monitored metrics per component.

Resolution – To stay under the limit, remove metrics that are not necessary.

No SAP metrics appear after the onboarding process

Use the following information to find out why SAP metrics don't appear on the dashboard after the onboarding process. The first step is to troubleshoot why the SAP metrics don't appear using the AWS Management Console or Exporter logs from an Amazon EC2 instance. Next, review the error output to find a resolution.

Troubleshoot why SAP metrics don't appear after onboarding

You can use the AWS Management Console or exporter logs from an Amazon EC2 instance for troubleshooting.

AWS Management Console

Troubleshoot no SAP metrics appear after onboarding using the console

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the left navigation pane, choose **State Manager**.

3. Under **Associations**, check the status of the document `AWSEC2-ApplicationInsightsCloudwatchAgentInstallAndConfigure`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.
4. Under **Associations**, check the status of the document `AWS-ConfigureAWSPackage`. If the status is `Failed`, under **Execution id**, select the failed id and view the output.

Exporter logs from Amazon EC2 instance

Troubleshoot no SAP metrics appear after onboarding using exporter logs

1. Connect to the Amazon EC2 instance where your SAP HANA database is running.
2. Find the correct naming convention for `WORKLOAD_SHORT_NAME` using the following command. You will use this short name in the following two steps.

```
sudo systemctl | grep exporter
```

> **Note**
>
> Application Insights adds a suffix, `WORKLOAD_SHORT_NAME` to the service name depending on the workload that is running. The short names for SAP HANA single node, multiple nodes, and high availability deployments are `HANA_SN`, `HANA_MN`, and `HANA_HA`.

3. To check for errors in the exporter manager service logs, run the following command replacing `WORKLOAD_SHORT_NAME` with the short name you found in Step 2.

```
sudo journalctl -e --unit=prometheus-
hanadb_exporter_manager_WORKLOAD_SHORT_NAME.service
```

4. If the exporter manager service logs do not show an error, check for errors in the exporter service logs by running the following command.

```
sudo journalctl -e --unit=prometheus-hanadb_exporter_WORKLOAD_SHORT_NAME.service
```

Resolving the common root causes for SAP metrics not appearing after onboarding

The following examples describe how to resolve the common root causes of SAP metrics not appearing after onboarding.

- The output shows the following error.

```
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-
cloudwatch-agent.d/default ...
Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/
amazon-cloudwatch-agent.d/ssm_AmazonCloudWatch-ApplicationInsights-
SSMParameterForTESTCWEC2INSTANCEi0d88867f1f3e36285.tmp ...
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Failed to merge multiple json config files.
2023/11/30 22:25:17 Under path : /metrics/append_dimensions | Error : Different
 values are specified for append_dimensions
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/disk | Error : Different
 values are specified for disk
2023/11/30 22:25:17 Under path : /metrics/metrics_collected/mem | Error : Different
 values are specified for mem
2023/11/30 22:25:17 Configuration validation first phase failed. Agent version: 1.0.
 Verify the JSON input is only using features supported by this version.
```

  Resolution – Application Insights is trying to configure the same metrics that are pre-configured as part of the existing CloudWatch agent configuration file. Remove the existing files under `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/` or remove the metrics that are causing the conflict from the existing CloudWatch agent configuration file.

- The output shows the following error.

```
Unable to find a host with system database, for more info rerun using -v
```

  Resolution – The username, password, or database port may be incorrect. Verify that the username, password, and port are valid, then re-run the onboarding process.

- The output shows the following error.

```
This hdbcli installer is not compatible with your Python interpreter
```

  Resolution – Upgrade pip3 and wheel as shown in the following example for Python 3.6.

```
python3.6 -m pip install --upgrade pip setuptools wheel
```

- The output shows the following error.

```
Unable to install hdbcli using pip3. Please try to install it
```

Resolution – Ensure that you have followed the `hdbclient` prerequisites or install `hdbclient` manually under pip3.

- The output shows the following error.

```
Package 'boto3' requires a different Python: 3.6.15 not in '>= 3.7'
```

Resolution – Python 3.8 or higher is required for this operating system version. Check the Python 3.8 prerequisites and install it.

- The output shows one of the following installation errors.

```
Can not execute `setup.py` since setuptools is not available in the build environment
```

or

```
[SSL: CERTIFICATE_VERIFY_FAILED]
```

Resolution – Install Python using SUSE Linux commands as shown in the following example. The following example installs the latest version of Python 3.8.

```
wget https://www.python.org/ftp/python/3.8.<LATEST_RELEASE>/
Python-3.8.<LATEST_RELEASE>.tgz
tar xf Python-3.*
cd Python-3.*/
sudo zypper install make gcc-c++ gcc automake autoconf libtool
sudo zypper install zlib-devel
sudo zypper install libopenssl-devel libffi-devel
./configure --with-ensurepip=install
sudo make
sudo make install
sudo su
python3.8 -m pip install --upgrade pip setuptools wheel
```

Tutorial: Set up monitoring for SAP NetWeaver

This tutorial demonstrates how to configure Amazon CloudWatch Application Insights to set up monitoring for SAP NetWeaver. You can use CloudWatch Application Insights automatic dashboards to visualize problem details, accelerate troubleshooting, and reduce mean time to resolution (MTTR) for your SAP NetWeaver application servers.

CloudWatch Application Insights for SAP NetWeaver topics

- Supported environments
- Supported operating systems
- Features
- Prerequisites
- Set up your SAP NetWeaver application servers for monitoring
- Manage monitoring of your SAP NetWeaver application servers
- View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights
- Troubleshooting Application Insights for SAP NetWeaver

Supported environments

CloudWatch Application Insights supports the deployment of AWS resources for the following systems and patterns.

- **SAP NetWeaver Standard System Deployment.**
- **SAP NetWeaver Distributed deployments on multiple Amazon EC2 instances.**
- **Cross-AZ SAP NetWeaver high availability setup** – SAP NetWeaver with high availability configured across two Availability Zones using SUSE/RHEL clustering.

Supported operating systems

CloudWatch Application Insights for SAP NetWeaver is supported on the following operating systems:

- Oracle Linux 8

- Red Hat Enterprise Linux 7.6
- Red Hat Enterprise Linux 7.7
- Red Hat Enterprise Linux 7.9
- Red Hat Enterprise Linux 8.1
- Red Hat Enterprise Linux 8.2
- Red Hat Enterprise Linux 8.4
- Red Hat Enterprise Linux 8.6
- SUSE Linux Enterprise Server 15 for SAP
- SUSE Linux Enterprise Server 15 SP1 for SAP
- SUSE Linux Enterprise Server 15 SP2 for SAP
- SUSE Linux Enterprise Server 15 SP3 for SAP
- SUSE Linux Enterprise Server 15 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP4 for SAP
- SUSE Linux Enterprise Server 12 SP5 for SAP
- SUSE Linux Enterprise Server 15 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP1 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP2 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP3 except High Availability patterns
- SUSE Linux Enterprise Server 15 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP4 except High Availability patterns
- SUSE Linux Enterprise Server 12 SP5 except High Availability patterns

Features

CloudWatch Application Insights for SAP NetWeaver 7.0x–7.5x (including ABAP Platform) provides the following features:

- Automatic SAP NetWeaver workload detection
- Automatic SAP NetWeaver alarm creation based on static thresholds
- Automatic SAP NetWeaver log pattern recognition
- Health dashboard for SAP NetWeaver

- Problem dashboard for SAP NetWeaver

Prerequisites

You must perform the following prerequisites to configure SAP NetWeaver with CloudWatch Application Insights:

- **AWS Systems Manager enablement** – Install SSM Agent on your Amazon EC2 instances, and enable the instances for SSM. For information about how to install the SSM Agent, see Setting up AWS Systems Manager in the *AWS Systems Manager User Guide*.
- **Amazon EC2 instance roles** – You must attach the following Amazon EC2 instance roles to configure your SAP NetWeaver monitoring.
  - You must attach the `AmazonSSMManagedInstanceCore` role to enable Systems Manager. For more information, see AWS Systems Manager identity-based policy examples.
  - You must attach the `CloudWatchAgentServerPolicy` policy to enable instance metrics and logs to be emitted through CloudWatch. For more information, see Create IAM roles and users for use with CloudWatch agent.
- **AWS resource groups** – You must create a resource group that includes all of the associated AWS resources used by your application stack to onboard your applications to CloudWatch Application Insights. This includes Amazon EC2 instances, Amazon EFS, and Amazon EBS volumes running your SAP NetWeaver application servers. If there are multiple SAP NetWeaver systems per account, we recommend that you create one resource group that includes the AWS resources for each SAP NetWeaver system. For more information about creating resource groups, see the *AWS Resource Groups and Tags User Guide*.
- **IAM permissions** – For users who don't have administrative access, you must create an AWS Identity and Access Management (IAM) policy that allows Application Insights to create a service-linked role and attach it to the user's identity. For more information about how to create the IAM policy, see IAM policy.
- **Service-linked role** – Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is created for you when you create your first Application Insights application in the Application Insights console. For more information, see Using service-linked roles for CloudWatch Application Insights.
- **Amazon CloudWatch agent** – Application Insights installs and configures the CloudWatch agent. If you have CloudWatch agent installed, Application Insights retains your configuration. To avoid a merge conflict, remove the configuration of resources that you want to use in Application

Insights from the existing CloudWatch agent configuration file. For more information, see Manually create or edit the CloudWatch agent configuration file.

Set up your SAP NetWeaver application servers for monitoring

Use the following steps to set up monitoring for your SAP NetWeaver application servers.

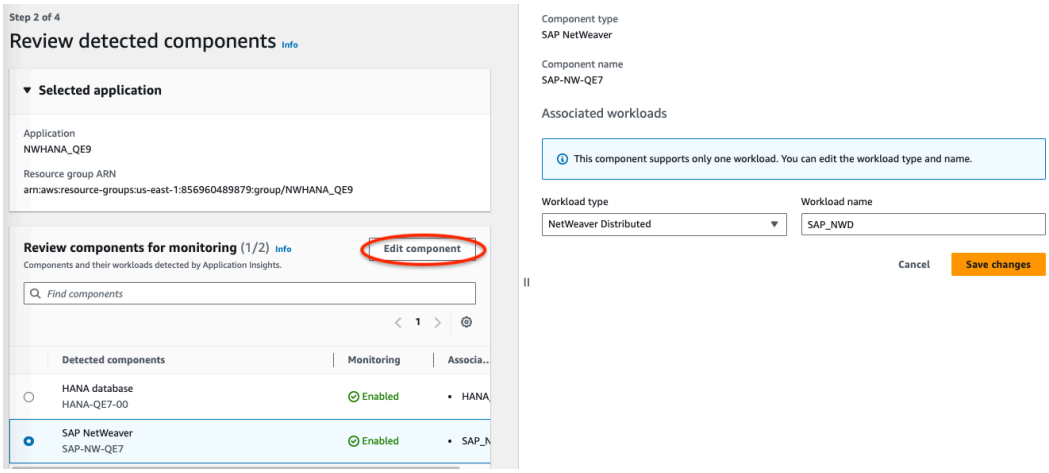
To set up monitoring

1. Open the CloudWatch console.
2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application. In the upper right-hand corner, select **Add an application**.
4. On the **Specify application details** page, from the dropdown list under **Resource group**, select the AWS resource group you created that contains your SAP NetWeaver resources. If you haven't created a resource group for your application, you can create one by choosing **Create new resource group** under the **Resource group** dropdown list.
5. Under **Automatic monitoring of new resources**, select the check box to allow Application Insights to automatically monitor the resources that are added to the application's resource group after onboarding.
6. Under **Monitor EventBridge events**, select the check box to integrate Application Insights monitoring with CloudWatch Events to get insights from Amazon EBS, Amazon EC2, AWS CodeDeploy, Amazon ECS, AWS Health APIs and notifications, Amazon RDS, Amazon S3, and AWS Step Functions.
7. Under **Integrate with AWS Systems Manager OpsCenter**, select the check box next to **Generate AWS Systems Manager OpsCenter OpsItems for remedial actions** to view and get notifications when problems are detected for the selected applications. To track the operations that are performed to resolve operational work items, called OpsItems, that are related to your AWS resources, provide an SNS topic ARN.
8. You can optionally enter tags to help you identify and organize your resources. CloudWatch Application Insights supports both tag-based and CloudFormation stack-based resource groups, with the exception of Application Auto Scaling groups. For more information, see Tag Editor in the *AWS Resource Groups and Tags User Guide*.
9. To review detected components, choose **Next**.

10. On the **Review detected components** page, the monitored components and their workloads automatically detected by CloudWatch Application Insights are listed.

    - To edit the workload type and name, choose **Edit component**.

    > **Note**
    >
    > Components that contain a detected NetWeaver Distributed or NetWeaver High Availability workload support only one workload on a component.



11. Choose **Next**.
12. On the **Specify component details** page, choose **Next**.
13. Review your application monitoring configuration, then choose **Submit**.
14. The application details page opens, where you can view the **Application summary**, **Dashboard**, **Components**, and **Workloads**. You can also view the **Configuration history**, **Log patterns**, and any **Tags** that you have created. After you submit your application, CloudWatch Application Insights deploys all of the metrics and alarms for your SAP NetWeaver system, which can take up to an hour.

Manage monitoring of your SAP NetWeaver application servers

Use the following steps to manage monitoring of your SAP NetWeaver application servers.

To manage monitoring

1. Open the CloudWatch console.

2. From the left navigation pane, under **Insights**, select **Application Insights**.
3. Choose the **List view** tab.
4. The **Application Insights** page displays the list of applications that are monitored with Application Insights, and the monitoring status for each application.
5. Select your application.
6. Choose the **Components** tab.
7. Under **Monitored components**, select the radio button next to the component name. Then, select **Manage monitoring**.
8. Under **Instance logs**, you can update the existing log path, log pattern set, and log group name. In addition, you can add up to three additional **Application logs**.
9. Under **Metrics**, you can select the SAP NetWeaver metrics according to your requirements. SAP NetWeaver metric names are prefixed with `sap`. You can add up to 40 metrics per component.
10. Under **Custom alarms**, you can add additional alarms to be monitored by CloudWatch Application Insights.
11. Review your application monitoring configuration and choose **Save**. When you submit your configuration, your account updates all of the metrics and alarms for your SAP NetWeaver systems.

View and troubleshoot SAP NetWeaver problems detected by CloudWatch Application Insights

The following sections provide steps to help you resolve common troubleshooting scenarios that occur when you configure monitoring for SAP NetWeaver on Application Insights.

Troubleshooting topics

- SAP NetWeaver database connectivity issues
- SAP NetWeaver application availability issues

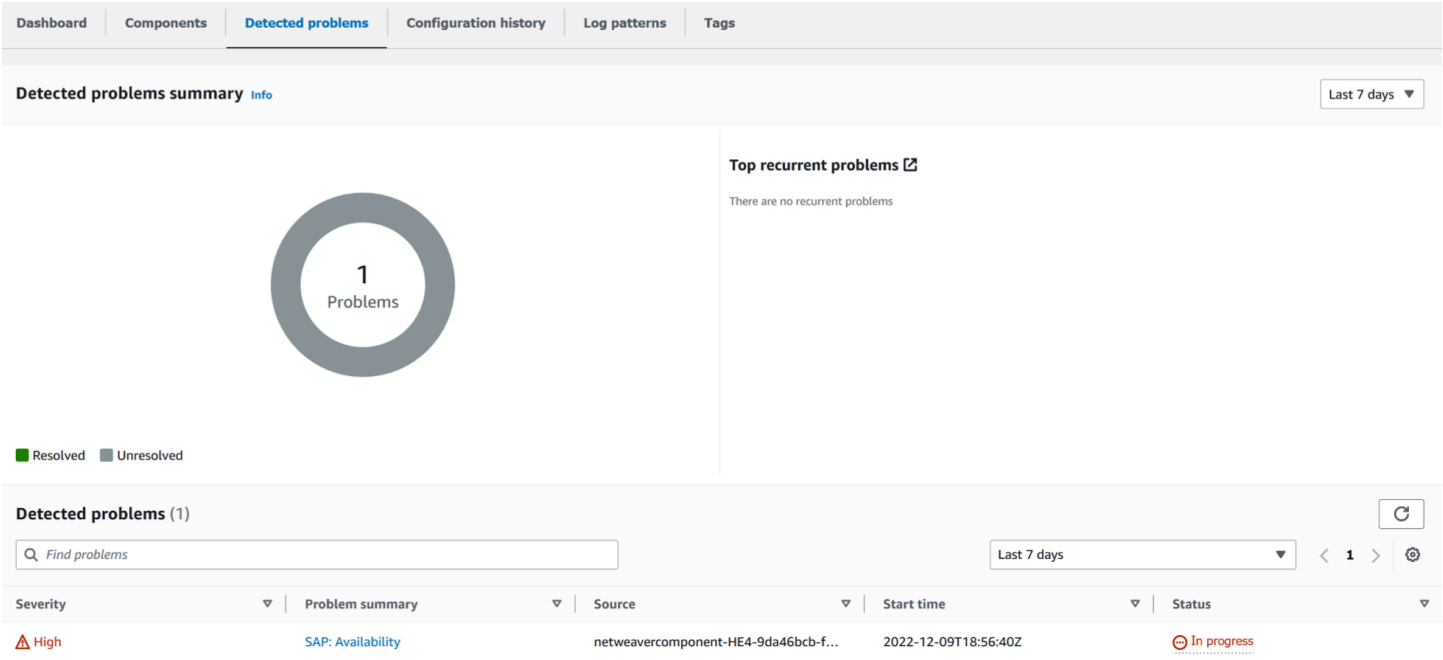
SAP NetWeaver database connectivity issues

Description

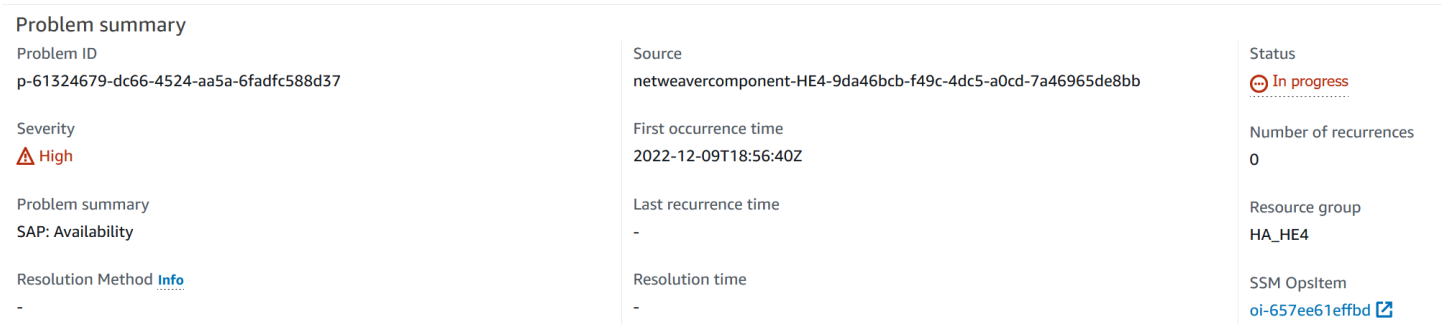
Your SAP NetWeaver application experiences database connectivity issues.

Cause

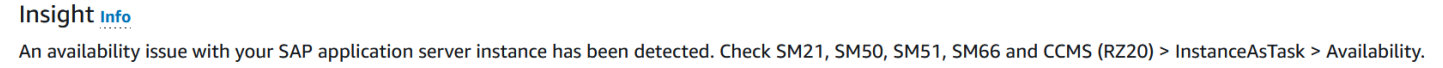
You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.



In the following example, under **Problem summary**, SAP: Availability is the issue.

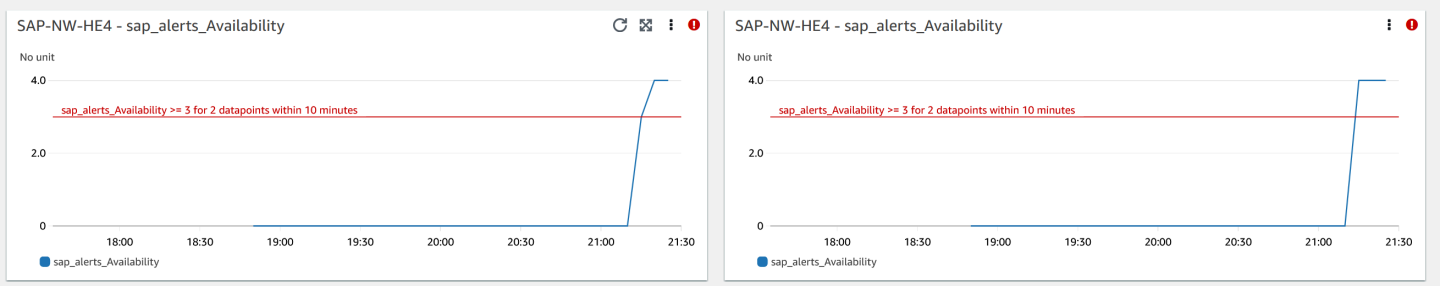


Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.

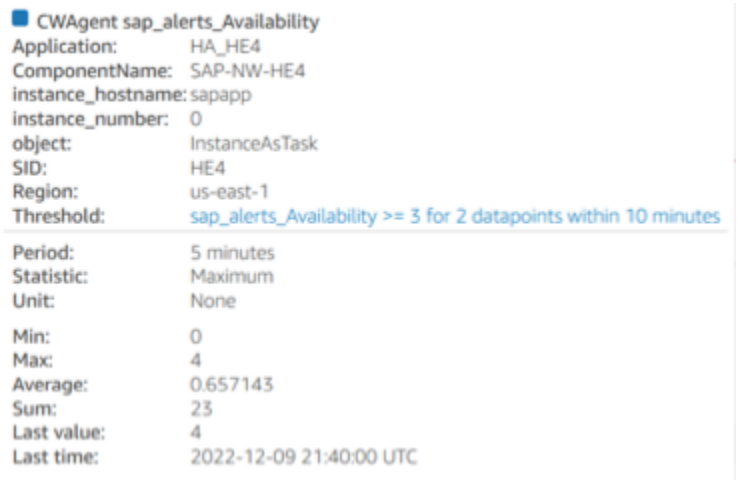


On the same problem dashboard, you can view related logs and metrics that problem detection has grouped together to help you isolate the cause of the error. The `sap_alerts_Availability` metric tracks the availability of the SAP NetWeaver system over time. You can use historical tracking to correlate when the metric initiated an error state or breached the alarm threshold. In

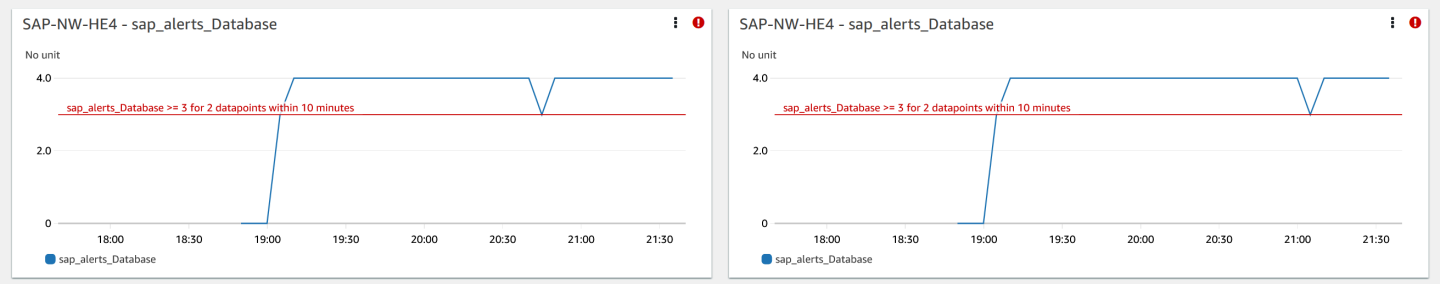
the following example, there is an availability issue with the SAP NetWeaver system. The example shows two alarms because there are two SAP application server instances and an alarm was created for each instance.



For more information about each alarm, hover over the `sap_alerts_Availability` metric name.



In the following example, the `sap_alerts_Database` metric shows that the database layer has an issue or a failure. This alarm indicates that SAP NetWeaver had issues connecting to or communicating with its database.



Since the database is a key resource for SAP NetWeaver, you may get many related alarms when the database has an issue or failure. In the following example, the

`sap_alerts_FrontendResponseTime` and `sap_alerts_LongRunners` metrics are initiated because the database is not available.



Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to the CloudWatch alarms. After the error conditions are fixed, the alarm is resolved when the alarms and logs are recovered. When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard.

In the following example, the SAP Availability issue is resolved.



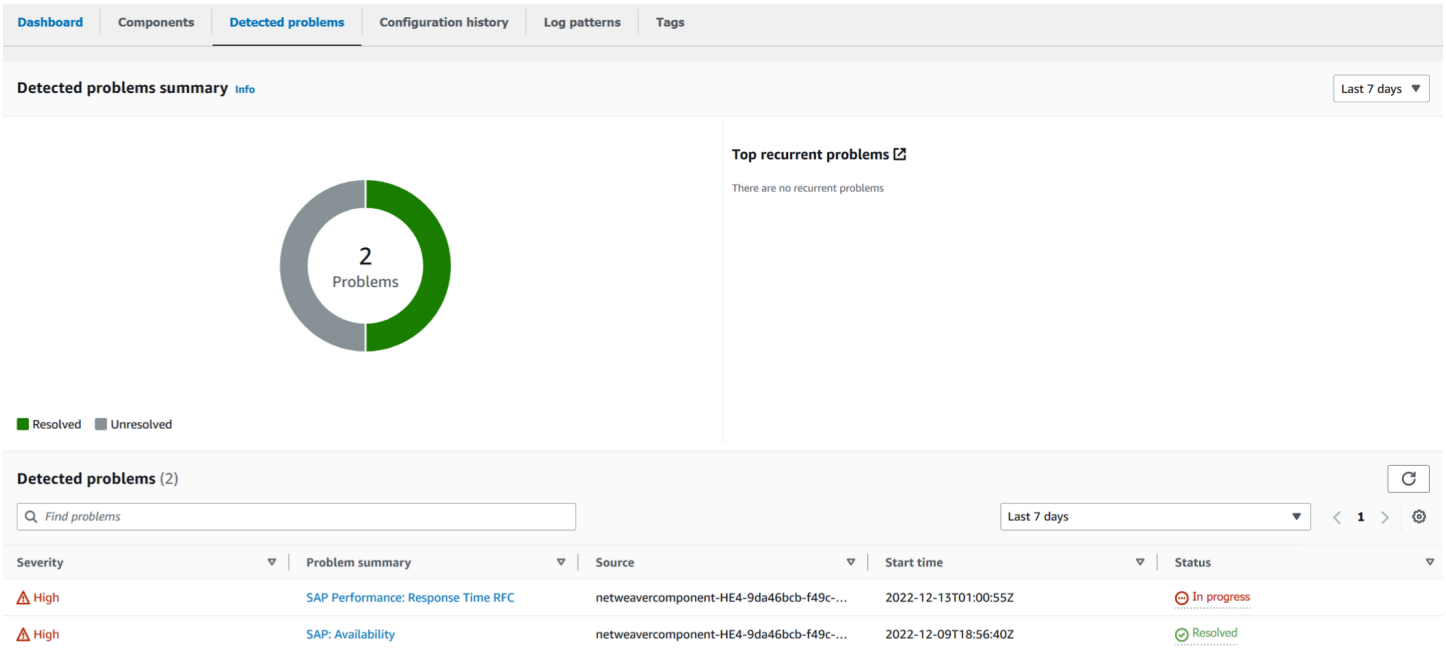
SAP NetWeaver application availability issues

Description

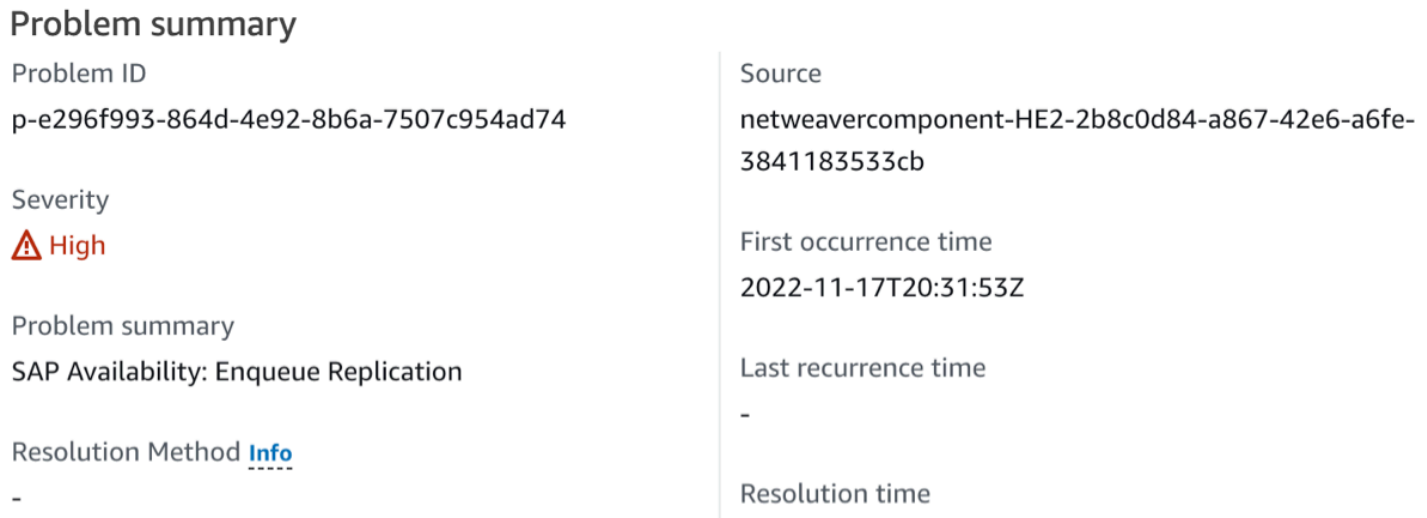
Your SAP NetWeaver High Availability Enqueue replication stopped working.

Cause

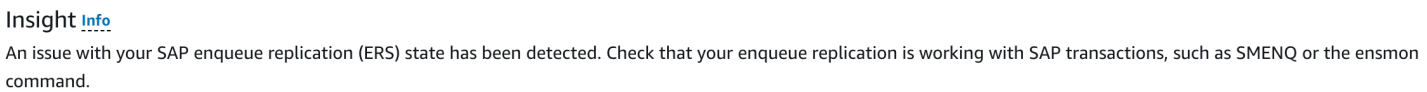
You can identify the connectivity issue by going to the CloudWatch Application Insights console and checking the SAP NetWeaver Application Insights problem dashboard. Select the link under **Problem summary** to see the specific issue.



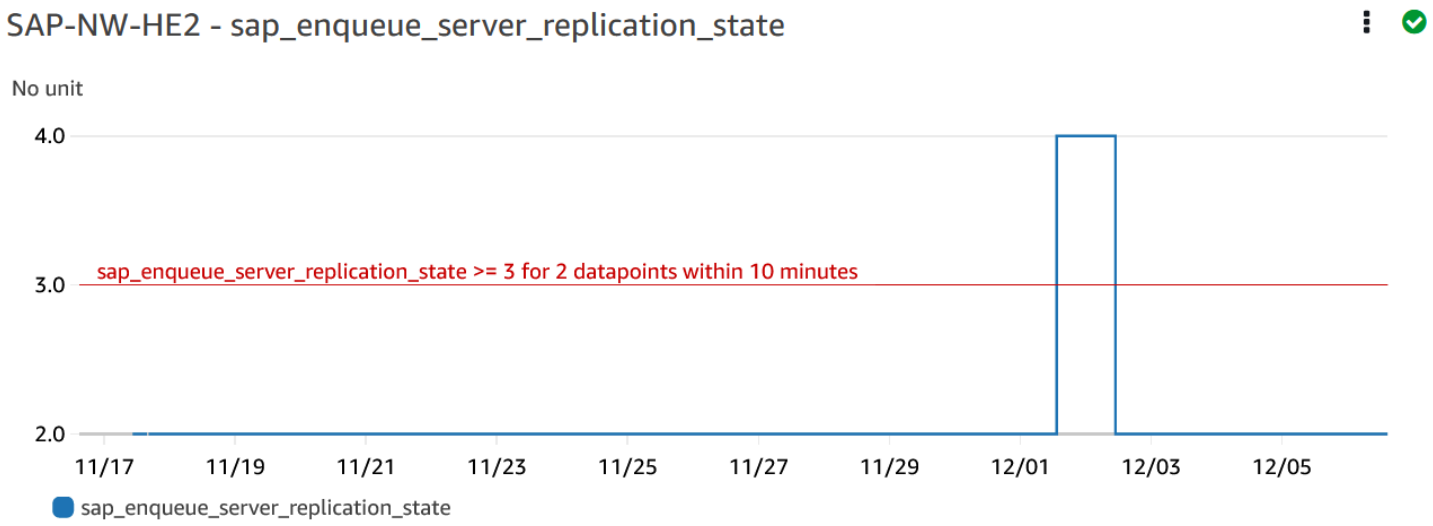
In the following example, under **Problem summary**, High Availability Enqueue Replication is the issue.



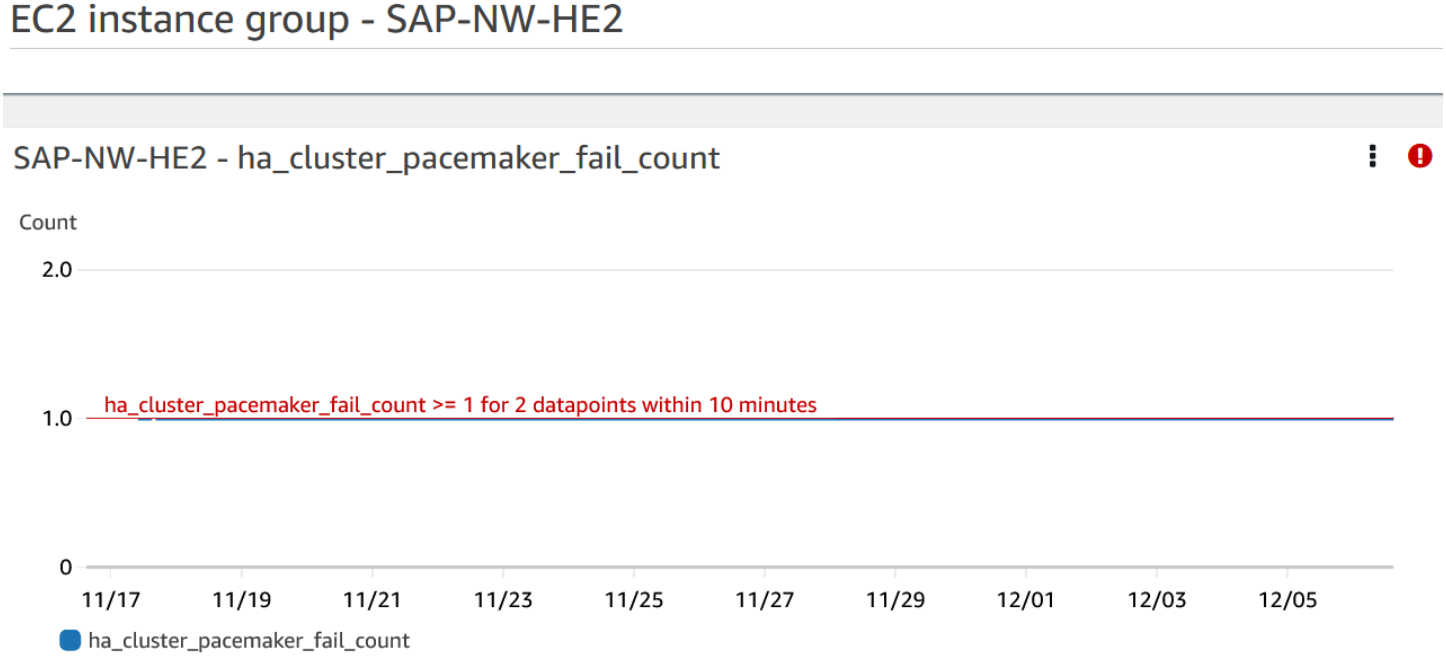
Immediately following the **Problem summary**, the **Insight** section provides more context about the error and where you can get more information about the causes of the issue.



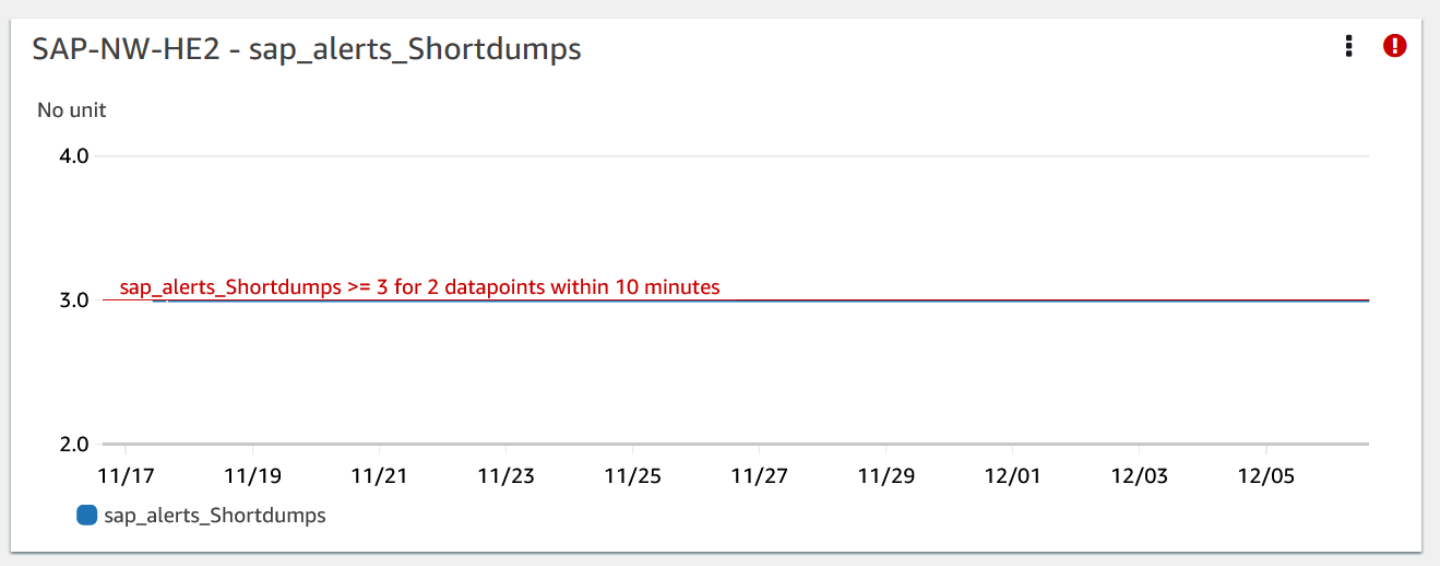
The following example shows the problem dashboard where you view logs and metrics which are grouped to help you isolate the causes of the error. The `sap_enqueue_server_replication_state` metric tracks the value over time. You can use historical tracking to correlate when the metric initiated an error state or breached the alarm threshold.



In the following example, the `ha_cluster_pacemaker_fail_count` metric shows that the high availability pacemaker cluster experienced a resource failure. The specific pacemaker resources that had a fail count greater than or equal to one are identified in the component dashboard.

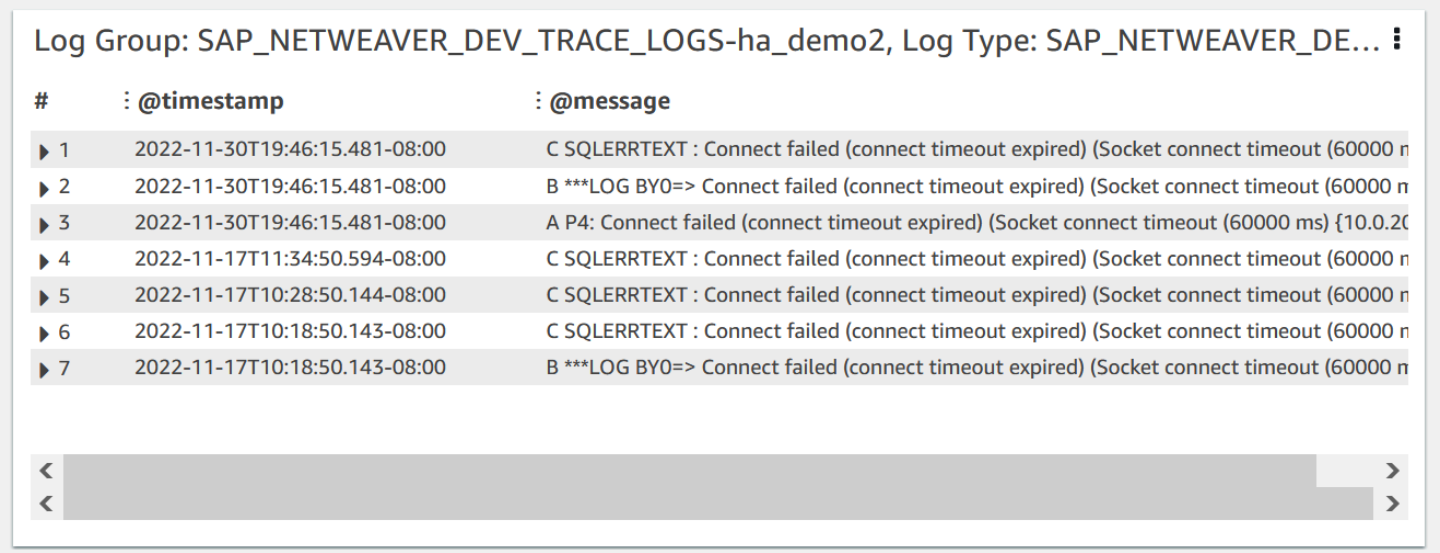


The following example shows the `sap_alerts_Shortdumps` metric, which indicates that the SAP application performance was reduced when the problem was detected.

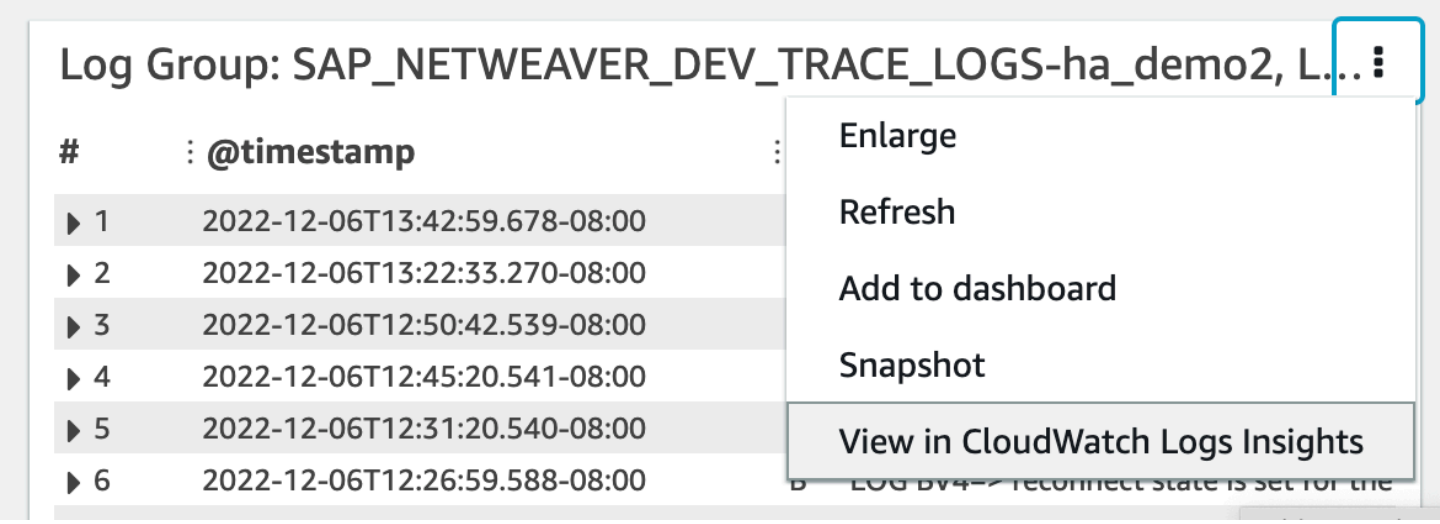


Logs

The log entries are helpful to get a better understanding of issues that occurred at the SAP NetWeaver layer when the problem was detected. The log group widget in the problem dashboard shows the specific time of the issue.



To see detailed information about the logs, select the three vertical dots in the upper-right corner, and select **View in CloudWatch Logs Insights**.

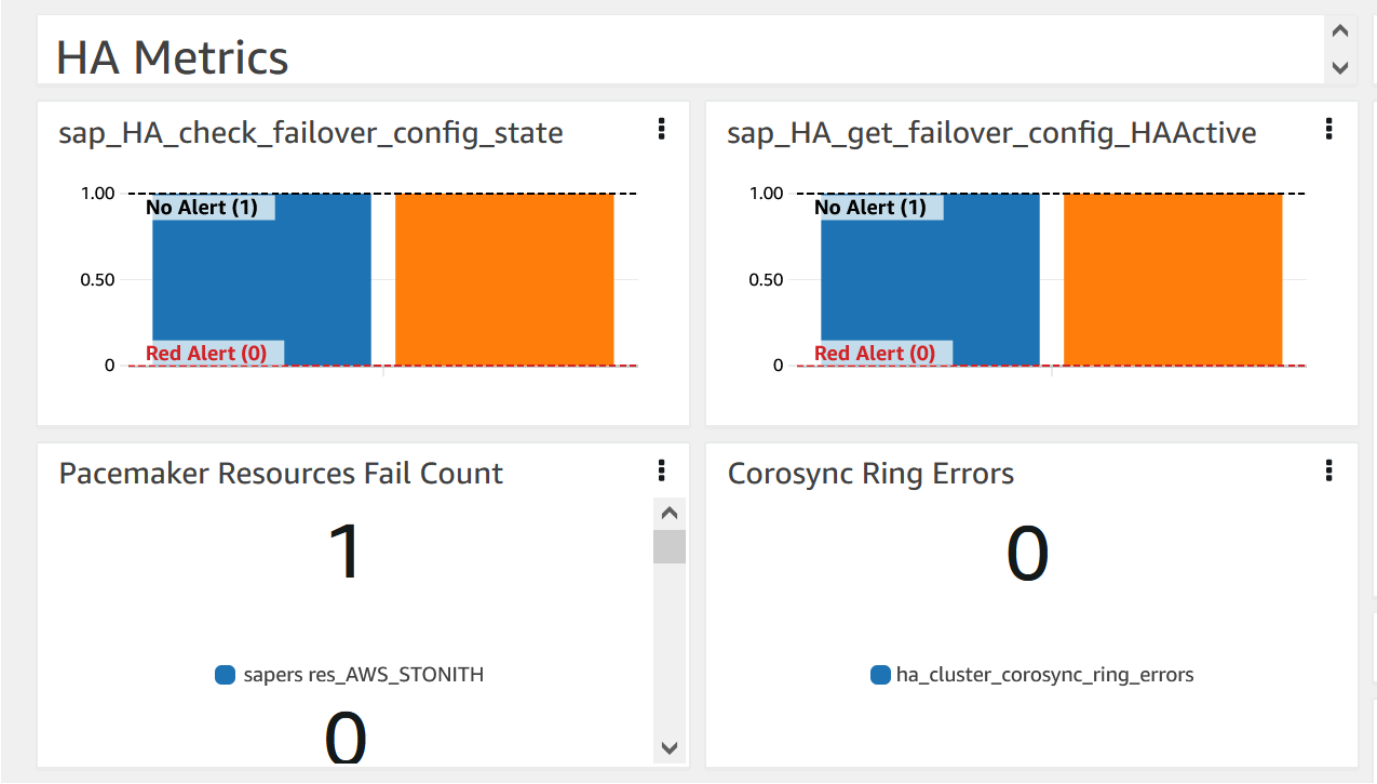


Use the following steps to get more information about the metrics and alarms displayed in the problem dashboard.

To get more information about metrics and alarms

1. Open the CloudWatch console.
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Select the **Components** tab. Then, select the SAP NetWeaver component about which you want to get more information.

   The following example shows the **HA Metrics** section with the `ha_cluster_pacemaker_fail_count` metric that was displayed in the problem dashboard.



Resolution

Application Insights monitors the detected problem hourly. If there are no new related log entries in your SAP NetWeaver log files, the older log entries will be treated as resolved. You must fix any error conditions related to this problem.

For the `sap_alerts_Shortdumps` alarm, you must resolve the alert in the SAP NetWeaver system by using transaction code `RZ20 # R3Abap # Shortdumps` to navigate to the CCMS alert. For more information about CCMS alerts, see the SAP website. Resolve all of the CCMS alerts in the Shortdumps tree. After all of the alerts are resolved in the SAP NetWeaver system, CloudWatch no longer reports the metric in an alarm state.

When all of the CloudWatch log errors and alarms are resolved, Application Insights stops detecting errors and the problem is automatically resolved within an hour. We recommend that you resolve all log error conditions and alarms so that you have the latest problems on the problem dashboard. In the following example, the SAP Netweaver High Availability Enqueue Replication problem is resolved.

Severity	Problem summary	Source	Start time	Status
High	SAP Availability: Enqueue Replication	netweavercomponent-HE2-2b8c0...	2022-12-08T20:01:43Z	Resolved

Troubleshooting Application Insights for SAP NetWeaver

This section provides steps to help you resolve common errors returned by the Application Insights dashboard.

Unable to add more than 60 monitor metrics

Error returned: `Component cannot have more than 60 monitored metrics.`

Root cause: `The current metric limit is 60 monitor metrics per component.`

Resolution: Remove metrics that are not necessary to adhere to the limit.

SAP metrics do not appear on the dashboard after the onboarding process

Root cause: Component Dashboard uses a five minute metric period to aggregate the data points.

Resolution: All metrics should show up on the dashboard after five minutes.

SAP metrics and alarms don't appear on the dashboard

Use the following steps to identify why SAP metrics and alarms don't appear on the dashboard after the onboarding process.

To identify the issue with metrics and alarms

1. Open the CloudWatch console.
2. In the left navigation pane, under **Insights**, select **Application Insights**. Then, choose the **List view** tab, and select your application.
3. Choose the **Configuration history** tab.
4. If you see missing metrics datapoints, check for errors related to `prometheus-sap_host_exporter`.
5. If you don't find an error in the previous step, Connect to your Linux instance. For High Availability deployments, connect to the primary cluster Amazon EC2 instance.
6. In your instance, verify that the exporter is running by using the following command. The default port is `9680`. If you are using a different port, replace `9680` with the port you are using.

   ```
   curl localhost:9680/metrics
   ```

   If no data is returned, then the exporter failed to start.

7. To find the correct naming convention to use for WORKLOAD_SHORT_NAME in the next two steps, run the following command.

   > **Note**
   >
   > Application Insights adds a suffix, WORKLOAD_SHORT_NAME, to the service name depending on the workload that is running. The short names for NetWeaver Distributed, Standard, and High Availability deployments are SAP_NWD, SAP_NWS, and SAP_NWH.

   ```
   sudo systemctl | grep exporter
   ```

8. To check for errors in the exporter service logs, run the following command:

   ```
   sudo journalctl -e --unit=prometheus-sap_host_exporter_WORKLOAD_SHORT_NAME.service
   ```

9. To check for errors in the exporter manager service logs, run the following command:

   ```
   sudo journalctl -e --unit=prometheus-
   sap_host_exporter_manager_WORKLOAD_SHORT_NAME.service
   ```

   > **Note**
   >
   > This service should be up and running at all times.

   If this command does not return an error, continue to the next step.

10. To manually start the exporter, run the following command. Then, check the exporter output.

    ```
    sudo /opt/aws/sap_host_exporter/sap_host_exporter
    ```

    You can exit the exporter process after you check for errors.

Root cause: There are several possible causes for this issue. A common cause is that the exporter is not able to connect to one of the application server instances.

Resolution

Use the following steps to connect the exporter to the application server instances. You will verify that the SAP application instance is running and use SAPControl to connect to the instance.

To connect the exporter to the application server instances

1. In your Amazon EC2 instance, run the following command to verify that the SAP application is running.

   ```
   sapcontrol -nr <App_InstNo> -function GetProcessList
   ```

2. You must establish a working SAPControl connection. If the SAPControl connection doesn't work, find the root cause of the issue on the relevant SAP application instance.
3. To manually start the exporter after you fix the SAP Control connection issue, run the following command:

   ```
   sudo systemctl start prometheus-sap_host_exporter.service
   ```

4. If you can't resolve the SAPControl connection issue, use the following procedure as a temporary fix.

   a. Open the AWS Systems Manager console.

   b. From the left navigation pane, choose **State Manager**.

   c. Under **Associations** search for the SAP NetWeaver system's association.

   ```
   Association Name: Equal: AWS-ApplicationInsights-
   SSMSAPHostExporterAssociationForCUSTOMSAPNW<SID>-1
   ```

   d. Select the **Association id**.

   e. Choose the **Parameters** tab and remove the application server number from **additionalArguments**.

   f. Choose **Apply Association Now**.

   > **Note**
   >
   > This is a temporary fix. If updates are made to the component's monitoring configurations, the instance will be added back.

View and troubleshoot problems detected by Amazon CloudWatch Application Insights

The topics in this section provide detailed information about the detected problems and insights displayed by Application Insights. It also provides suggested resolutions for detected issues with your account or your configuration.

Troubleshooting topics

- CloudWatch console overview
- Application Insights problem summary page
- CloudWatch agent merge conflict failures
- High CPU usage from CloudWatch agent log processing
- Alarms are not created
- Feedback
- Configuration errors

CloudWatch console overview

An overview of problems that impact your monitored applications can be found under the CloudWatch Application Insights pane in the overview page of the CloudWatch console.

The CloudWatch Application Insights overview pane displays the following:

- The severity of the problems detected: High/Medium/Low
- A short summary of the problem
- The problem source
- The time the problem started
- The resolution status of the problem
- The affected resource group

To view the details of a specific problem, under **Problem Summary**, select the description of the problem. A detailed dashboard displays insights into the problem and related metric anomalies and snippets of log errors. You can provide feedback on the relevance of the insight by selecting whether it is useful.

If a new resource is detected that is not configured, the problem summary description takes you to the **Edit configuration** wizard to configure your new resource. You can view or edit your resource group configuration by choosing **View/edit configuration** in the upper right-hand corner of the detailed dashboard.

To return to the overview, choose **Back to overview**, which is next to the CloudWatch Application Insights detailed dashboard header.

Application Insights problem summary page

Application Insights problem summary page

CloudWatch Application Insights provides the following information about detected problems on the problem summary page:

- A short summary of the problem
- The start time and date of the problem
- The problem severity: High/Medium/Low
- The status of the detected problem: In-progress/Resolved
- Insights: Automatically generated insights on the detected problem and possible root cause
- Feedback on insights: Feedback you have provided about the usefulness of the insights generated by CloudWatch Application Insights
- Related observations: A detailed view of the metric anomalies and error snippets of relevant logs related to the problem across various application components

CloudWatch agent merge conflict failures

CloudWatch Application Insights installs and configures the CloudWatch agent on customer instances. This includes creation of a CloudWatch agent configuration file with configurations for metrics or logs. A merge conflict can occur if a customer's instance already has a CloudWatch agent configuration file with different configurations defined for the same metrics or logs. To resolve the merge conflict, use the following steps:

1. Identify the CloudWatch agent configuration files on your system. For more information about the file locations, see CloudWatch agent files and locations.

2. Remove the resource configurations that you want to use in Application Insights from the existing CloudWatch agent configuration file. If you want to only use Application Insights configurations, delete the existing CloudWatch agent configuration files.

High CPU usage from CloudWatch agent log processing

CloudWatch Application Insights installs and configures the CloudWatch agent on customer instances. If an Amazon EC2 instance is configured with log paths that have large amounts of log data, the instance might experience increases in CPU usage while the CloudWatch agent processes the logs. To reduce CPU usage, remove the log path in the Amazon EC2 instance component configuration.

Alarms are not created

For some metrics, Application Insights predicts the alarm threshold based on previous data points for the metric. To enable this prediction, the following criteria must be met.

- **Recent data points** – There must be a minimum of 100 data points from the last 24 hours. The data points don't need to be continuous and can be scattered throughout the 24 hour time frame.
- **Historical data** – There must be a minimum of 100 data points spanning the time frame from 15 days before the current date to 1 day before the current date. The data points don't need to be continuous and can be scattered throughout the 15 day time frame.

> **Note**
>
> For some metrics, Application Insights delays the creation of alarms until the preceding conditions are met. In this case, you get a configuration history event that the metric lacks sufficient data points to establish the alarm threshold.

Feedback

Feedback

You can provide feedback on the automatically generated insights on detected problems by designating them useful or not useful. Your feedback on the insights, along with your application

diagnostics (metric anomalies and log exceptions), are used to improve the future detection of similar problems.

Configuration errors

CloudWatch Application Insights uses your configuration to create monitoring telemetries for the components. When Application Insights detects an issue with your account or your configuration, information is provided in the **Remarks** field of the Application summary about how to resolve the configuration issue for your application.

The following table shows suggested resolutions for specific remarks.

Remarks	Suggested resolution	Additional notes
The quota for CloudFormation has already been reached.	Application Insights creates one CloudFormation stack for each application to manage CloudWatch agent installation and configuration for all application components. By default, each AWS account can have 2000 stacks. See CloudFormation Limits. To resolve this, raise the limit for CloudFormation stacks.	n/a
No SSM instance role on the following instances.	For Application Insights to be able to install and configure CloudWatch agent on application instances, AmazonSSMManagedInstanceCore and CloudWatchAgentServerPolicy policies must be attached to the instance role.	Application Insights calls the SSM DescribeInstanceInformation API to get the list of instances with SSM permission. After the role is attached to the instance, it takes time for SSM to include the instance in the DescribeInstanceInformation result. Until SSM includes the instance in the result, NO_SSM_INSTANCE_ROLE

Remarks	Suggested resolution	Additional notes
		error remains present for the application.
New components may need configuration.	Application Insights detects that there are new components in the applicati on Resource Group. To resolve this, configure the new components accordingly.	n/a

Logs and metrics supported by Amazon CloudWatch Application Insights

The following lists show the supported logs and metrics for Amazon CloudWatch Application Insights.

CloudWatch Application Insights supports the following logs:

- Microsoft Internet Information Services (IIS) logs
- Error log for SQL Server on EC2
- Custom .NET application logs, such as Log4Net
- Windows Event logs, including Windows logs (System, Application, and Security) and Applications and Services log
- Amazon CloudWatch Logs for AWS Lambda
- Error log and slow log for RDS MySQL, Aurora MySQL, and MySQL on EC2
- Postgresql log for PostgreSQL RDS and PostgreSQL on EC2
- Amazon CloudWatch Logs for AWS Step Functions
- Execution logs and access logs (JSON, CSV, and XML, but not CLF) for API Gateway REST API stages
- Prometheus JMX exporter logs (EMF)
- Alert logs and listener logs for Oracle on Amazon RDS and Oracle on Amazon EC2
- Container logs routing from Amazon ECS containers to CloudWatch using `awslogs` log driver.

- Container logs routing from Amazon ECS containers to CloudWatch using FireLens container log router.
- Container logs routing from Amazon EKS or Kubernetes running on Amazon EC2 to CloudWatch using Fluent Bit or Fluentd log processor with Container Insights.
- SAP HANA trace and error logs
- HA Pacemaker logs
- SAP ASE server logs
- SAP ASE backup server logs
- SAP ASE Replication server logs
- SAP ASE RMA agent logs
- SAP ASE Fault Manager logs
- SAP NetWeaver dev trace logs
- Process metrics for Windows processes using proctstat plugin for CloudWatch agent
- Public DNS query logs for hosted zone
- Amazon Route 53 Resolver DNS query logs

CloudWatch Application Insights supports the following log classes:

- **Standard** – Amazon CloudWatch Application Insights requires that log groups are configured with the CloudWatch Logs Standard log class to enable monitoring.

CloudWatch Application Insights supports metrics for the following application components:

- Amazon Elastic Compute Cloud (EC2)
  - CloudWatch built-in metrics
  - CloudWatch agent metrics (Windows server)
  - CloudWatch agent process metrics (Windows server)
  - CloudWatch agent metrics (Linux server)
- Elastic Block Store (EBS)
- Amazon Elastic File System (Amazon EFS)
- Elastic Load Balancer (ELB)
- Application ELB

- Amazon EC2 Auto Scaling groups
- Amazon Simple Queue Server (SQS)
- Amazon Relational Database Service (RDS)
  - RDS Database instances
  - RDS Database clusters
- AWS Lambda function
- Amazon DynamoDB table
- Amazon S3 bucket
- AWS Step Functions
  - Execution-level
  - Activity
  - Lambda function
  - Service integration
  - Step Functions API
- API Gateway REST API stages
- SAP HANA
- SAP ASE
- SAP ASE High Availability on Amazon EC2
- SAP NetWeaver
- HA Cluster
- Java
- Amazon Elastic Container Service (Amazon ECS)
  - CloudWatch built-in metrics
  - Container Insights metrics
  - Container Insights Prometheus metrics
- Kubernetes on AWS
  - Container Insights metrics
  - Container Insights Prometheus metrics
- Amazon FSx
- Amazon VPC

- Amazon VPC NAT gateways
- Amazon Route 53 health check
- Amazon Route 53 hosted zone
- Amazon Route 53 Resolver endpoint
- AWS Network Firewall rule group
- AWS Network Firewall rule group association
- Metrics with data points requirements
  - AWS/ApplicationELB
  - AWS/AutoScaling
  - AWS/EC2
  - Elastic Block Store (EBS)
  - AWS/ELB
  - AWS/RDS
  - AWS/Lambda
  - AWS/SQS
  - AWS/CWAgent
  - AWS/DynamoDB
  - AWS/S3
  - AWS/States
  - AWS/ApiGateway
  - AWS/SNS
- Recommended metrics
- Performance Counter metrics

Amazon Elastic Compute Cloud (EC2)

CloudWatch Application Insights supports the following metrics:

Metrics

- CloudWatch built-in metrics
- CloudWatch agent metrics (Windows server)
- CloudWatch agent process metrics (Windows server)

- CloudWatch agent metrics (Linux server)

CloudWatch built-in metrics

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

CloudWatch agent metrics (Windows server)

.NET CLR Exceptions # of Exceps Thrown

.NET CLR Exceptions # of Exceps Thrown/Sec

.NET CLR Exceptions # of Filters/sec

.NET CLR Exceptions # of Finallys/sec

.NET CLR Exceptions Throw to Catch Depth/sec

.NET CLR Interop # of CCWs

.NET CLR Interop # of Stubs

.NET CLR Interop # of TLB exports/sec

.NET CLR Interop # of TLB imports/sec

.NET CLR Interop # of marshaling

.NET CLR Jit % Time in Jit

.NET CLR Jit Standard Jit Failures

.NET CLR Loading % Time Loading

.NET CLR Loading Rate of Load Failures

.NET CLR LocksAndThreads Contention Rate/sec

.NET CLR LocksAndThreads Queue Length/sec

.NET CLR Memory # Total Committed Bytes

.NET CLR Memory % Time in GC

.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time

.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/sec

.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/sec

APP_POOL_WAS Total Worker Process Ping Failures

ASP.NET Application Restarts

ASP.NET Applications % Managed Processor Time (estimated)

ASP.NET Applications Errors Total/Sec

ASP.NET Applications Errors Unhandled During Execution/sec

ASP.NET Applications Requests in Application Queue

ASP.NET Applications Requests/Sec

ASP.NET Request Wait Time

ASP.NET Requests Queued

HTTP Service Request Queues CurrentQueueSize

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Memory Pages/sec

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk Queue Length

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Full Scans/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:General Statistics Processes blocked

SQLServer:General Statistics User Connections

SQLServer:Latches Average Latch Wait Time (ms)

SQLServer:Locks Average Wait Time (ms)

SQLServer:Locks Lock Timeouts/sec

SQLServer:Locks Lock Waits/sec

SQLServer:Locks Number of Deadlocks/sec

SQLServer:Memory Manager Memory Grants Pending

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

W3SVC_W3WP File Cache Flushes

W3SVC_W3WP File Cache Misses

W3SVC_W3WP Requests/Sec

W3SVC_W3WP URI Cache Flushes

W3SVC_W3WP URI Cache Misses

Web Service Bytes Received/Sec

Web Service Bytes Sent/Sec

Web Service Connection attempts/sec

Web Service Current Connections

Web Service Get Requests/sec

Web Service Post Requests/sec

Bytes Received/sec

Normal Messages Queue Length/sec

Urgent Message Queue Length/sec

Reconnect Count

Unacknowledged Message Queue Length/sec

Messages Outstanding

Messages Sent/sec

Database Update Messages/sec

Update Messages/sec

Flushes/sec

Crypto Checkpoints Saved/sec

Crypto Checkpoints Restored/sec

Registry Checkpoints Restored/sec

Registry Checkpoints Saved/sec

Cluster API Calls/sec

Resource API Calls/sec

Cluster Handles/sec

Resource Handles/sec

CloudWatch agent process metrics (Windows server)

Process metrics are collected using the CloudWatch agent procstat plugin. Only Amazon EC2 instances running Windows workloads support process metrics.

procstat cpu_time_system

procstat cpu_time_user

procstat cpu_usage

procstat memory_rss

procstat memory_vms

procstat read_bytes

procstat write_bytes

.procstat read_count

procstat write_count

CloudWatch agent metrics (Linux server)

cpu_time_active

cpu_time_guest

cpu_time_guest_nice

cpu_time_idle

cpu_time_iowait

cpu_time_irq

cpu_time_nice

cpu_time_softirq

cpu_time_steal

cpu_time_system

cpu_time_user

cpu_usage_active

cpu_usage_guest

cpu_usage_guest_nice

cpu_usage_idle

cpu_usage_iowait

cpu_usage_irq

cpu_usage_nice

cpu_usage_softirq

cpu_usage_steal

cpu_usage_system

cpu_usage_user

disk_free

disk_inodes_free

disk_inodes_used

disk_used

disk_used_percent

diskio_io_time

diskio_iops_in_progress

diskio_read_bytes

diskio_read_time

diskio_reads

diskio_write_bytes

diskio_write_time

diskio_writes

mem_active

mem_available

mem_available_percent

mem_buffered

mem_cached

mem_free

mem_inactive

mem_used

mem_used_percent

net_bytes_recv

net_bytes_sent

net_drop_in

net_drop_out

net_err_in

net_err_out

net_packets_recv

net_packets_sent

netstat_tcp_close

netstat_tcp_close_wait

netstat_tcp_closing

netstat_tcp_established

netstat_tcp_fin_wait1

netstat_tcp_fin_wait2

netstat_tcp_last_ack

netstat_tcp_listen

netstat_tcp_none

netstat_tcp_syn_recv

netstat_tcp_syn_sent

netstat_tcp_time_wait

netstat_udp_socket

processes_blocked

processes_dead

processes_idle

processes_paging

processes_running

processes_sleeping

processes_stopped

processes_total

processes_total_threads

processes_wait

processes_zombies

swap_free

swap_used

swap_used_percent

Elastic Block Store (EBS)

CloudWatch Application Insights supports the following metrics:

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

Amazon Elastic File System (Amazon EFS)

CloudWatch Application Insights supports the following metrics:

BurstCreditBalance

PercentIOLimit

PermittedThroughput

MeteredIOBytes

TotalIOBytes

DataWriteIOBytes

DataReadIOBytes

MetadataIOBytes

ClientConnections

TimeSinceLastSync

StorageBytes

Throughput

PercentageOfPermittedThroughputUtilization

ThroughputIOPS

PercentThroughputDataReadIOByte

PercentThroughputDataWriteIOBytes

PercentageOfIOPSDataReadIOBytes

PercentageOfIOPSDataWriteIOBytes

AverageDataReadIOBytesSize

AverageDataWriteIOBytesSize

Elastic Load Balancer (ELB)

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

RequestCount

UnHealthyHostCount

Application ELB

CloudWatch Application Insights supports the following metrics:

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

Amazon EC2 Auto Scaling groups

CloudWatch Application Insights supports the following metrics:

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

StatusCheckFailed

StatusCheckFailed_Instance

StatusCheckFailed_System

Amazon Simple Queue Server (SQS)

CloudWatch Application Insights supports the following metrics:

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

Amazon Relational Database Service (RDS)

CloudWatch Application Insights supports the following metrics:

Metrics

- RDS Database instances
- RDS Database clusters

RDS Database instances

BurstBalance

CPUCreditBalance

CPUUtilization

DatabaseConnections

DiskQueueDepth

FailedSQLServerAgentJobsCount

FreeStorageSpace

FreeableMemory

NetworkReceiveThroughput

NetworkTransmitThroughput

ReadIOPS

ReadLatency

ReadThroughput

WriteIOPS

WriteLatency

WriteThroughput

RDS Database clusters

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

BufferCacheHitRatio

CPUUtilization

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

EngineUptime

FreeLocalStorage

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ResultSetCacheHitRatio

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

AWS Lambda function

CloudWatch Application Insights supports the following metrics:

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

Amazon DynamoDB table

CloudWatch Application Insights supports the following metrics:

SystemErrors

UserErrors

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

Amazon S3 bucket

CloudWatch Application Insights supports the following metrics:

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectBytesScanned

SelectBytesReturned

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS Step Functions

CloudWatch Application Insights supports the following metrics:

Metrics

- Execution-level
- Activity
- Lambda function
- Service integration
- Step Functions API

Execution-level

ExecutionTime

ExecutionThrottled

ExecutionsFailed

ExecutionsTimedOut

ExecutionsAborted

ExecutionsSucceeded

ExecutionsStarted

Activity

ActivityRunTime

ActivityScheduleTime

ActivityTime

ActivitiesFailed

ActivitiesHeartbeatTimedOut

ActivitiesTimedOut

ActivitiesScheduled

ActivitiesSucceeded

ActivitiesStarted

Lambda function

LambdaFunctionRunTime

LambdaFunctionScheduleTime

LambdaFunctionTime

LambdaFunctionsFailed

LambdaFunctionsTimedOut

LambdaFunctionsScheduled

LambdaFunctionsSucceeded

LambdaFunctionsStarted

Service integration

ServiceIntegrationRunTime

ServiceIntegrationScheduleTime

ServiceIntegrationTime

ServiceIntegrationsFailed

ServiceIntegrationsTimedOut

ServiceIntegrationsScheduled

ServiceIntegrationsSucceeded

ServiceIntegrationsStarted

Step Functions API

ThrottledEvents

ProvisionedBucketSize

ProvisionedRefillRate

ConsumedCapacity

API Gateway REST API stages

CloudWatch Application Insights supports the following metrics:

4XXError

5XXError

IntegrationLatency

Latency

CacheHitCount

CacheMissCount

SAP HANA

> **Note**
>
> CloudWatch Application Insights supports only single SID HANA environments. If multiple HANA SIDs are attached, monitoring will be set up for only the first detected SID.

CloudWatch Application Insights supports the following metrics:

hanadb_every_service_started_status

hanadb_daemon_service_started_status

hanadb_preprocessor_service_started_status

hanadb_webdispatcher_service_started_status

hanadb_compileserver_service_started_status

hanadb_nameserver_service_started_status

hanadb_server_startup_time_variations_seconds

hanadb_level_5_alerts_count

hanadb_level_4_alerts_count

hanadb_out_of_memory_events_count

hanadb_max_trigger_read_ratio_percent

hanadb_max_trigger_write_ratio_percent

hanadb_log_switch_wait_ratio_percent

hanadb_log_switch_race_ratio_percent

hanadb_time_since_last_savepoint_seconds

hanadb_disk_usage_highlevel_percent

hanadb_max_converter_page_number_count

hanadb_long_running_savepoints_count

hanadb_failed_io_reads_count

hanadb_failed_io_writes_count

hanadb_disk_data_unused_percent

hanadb_current_allocation_limit_used_percent

hanadb_table_allocation_limit_used_percent

hanadb_host_total_physical_memory_mb

hanadb_host_physical_memory_used_mb

hanadb_host_physical_memory_free_mb

hanadb_swap_memory_free_mb

hanadb_swap_memory_used_mb

hanadb_host_allocation_limit_mb

hanadb_host_total_memory_used_mb

hanadb_host_total_peak_memory_used_mb

hanadb_host_total_allocation_limit_mb

hanadb_host_code_size_mb

hanadb_host_shared_memory_allocation_mb

hanadb_cpu_usage_percent

hanadb_cpu_user_percent

hanadb_cpu_system_percent

hanadb_cpu_waitio_percent

hanadb_cpu_busy_percent

hanadb_cpu_idle_percent

hanadb_long_delta_merge_count

hanadb_unsuccessful_delta_merge_count

hanadb_successful_delta_merge_count

hanadb_row_store_allocated_size_mb

hanadb_row_store_free_size_mb

hanadb_row_store_used_size_mb

hanadb_temporary_tables_count

hanadb_large_non_compressed_tables_count

hanadb_total_non_compressed_tables_count

hanadb_longest_running_job_seconds

hanadb_average_commit_time_milliseconds

hanadb_suspended_sql_statements_count

hanadb_plan_cache_hit_ratio_percent

hanadb_plan_cache_lookup_count

hanadb_plan_cache_hit_count

hanadb_plan_cache_total_execution_microseconds

hanadb_plan_cache_cursor_duration_microseconds

hanadb_plan_cache_preparation_microseconds

hanadb_plan_cache_evicted_count

hanadb_plan_cache_evicted_microseconds

hanadb_plan_cache_evicted_preparation_count

hanadb_plan_cache_evicted_execution_count

hanadb_plan_cache_evicted_preparation_microseconds

hanadb_plan_cache_evicted_cursor_duration_microseconds

hanadb_plan_cache_evicted_total_execution_microseconds

hanadb_plan_cache_evicted_plan_size_mb

hanadb_plan_cache_count

hanadb_plan_cache_preparation_count

hanadb_plan_cache_execution_count

hanadb_network_collision_rate

hanadb_network_receive_rate

hanadb_network_transmit_rate

hanadb_network_packet_receive_rate

hanadb_network_packet_transmit_rate

hanadb_network_transmit_error_rate

hanadb_network_receive_error_rate

hanadb_time_until_license_expires_days

hanadb_is_license_valid_status

hanadb_local_running_connections_count

hanadb_local_idle_connections_count

hanadb_remote_running_connections_count

hanadb_remote_idle_connections_count

hanadb_last_full_data_backup_age_days

hanadb_last_data_backup_age_days

hanadb_last_log_backup_age_hours

hanadb_failed_data_backup_past_7_days_count

hanadb_failed_log_backup_past_7_days_count

hanadb_oldest_backup_in_catalog_age_days

hanadb_backup_catalog_size_mb

hanadb_hsr_replication_status

hanadb_hsr_log_shipping_delay_seconds

hanadb_hsr_secondary_failover_count

hanadb_hsr_secondary_reconnect_count

hanadb_hsr_async_buffer_used_mb

hanadb_hsr_secondary_active_status

hanadb_handle_count

hanadb_ping_time_milliseconds

hanadb_connection_count

hanadb_internal_connection_count

hanadb_external_connection_count

hanadb_idle_connection_count

hanadb_transaction_count

hanadb_internal_transaction_count

hanadb_external_transaction_count

hanadb_user_transaction_count

hanadb_blocked_transaction_count

hanadb_statement_count

hanadb_active_commit_id_range_count

hanadb_mvcc_version_count

hanadb_pending_session_count

hanadb_record_lock_count

hanadb_read_count

hanadb_write_count

hanadb_merge_count

hanadb_unload_count

hanadb_active_thread_count

hanadb_waiting_thread_count

hanadb_total_thread_count

hanadb_active_sql_executor_count

hanadb_waiting_sql_executor_count

hanadb_total_sql_executor_count

hanadb_data_write_size_mb

hanadb_data_write_time_milliseconds

hanadb_log_write_size_mb

hanadb_log_write_time_milliseconds

hanadb_data_read_size_mb

hanadb_data_read_time_milliseconds

hanadb_log_read_size_mb

hanadb_log_read_time_milliseconds

hanadb_data_backup_write_size_mb

hanadb_data_backup_write_time_milliseconds

hanadb_log_backup_write_size_mb

hanadb_log_backup_write_time_milliseconds

hanadb_mutex_collision_count

hanadb_read_write_lock_collision_count

hanadb_admission_control_admit_count

hanadb_admission_control_reject_count

hanadb_admission_control_queue_size_mb

hanadb_admission_control_wait_time_milliseconds

SAP ASE

CloudWatch Application Insights supports the following metrics:

asedb_database_availability

asedb_trunc_log_on_chkpt_enabled

asedb_last_db_backup_age_in_days

asedb_last_transaction_log_backup_age_in_hours

asedb_suspected_database

asedb_db_space_usage_percent

asedb_db_log_space_usage_percent

asedb_locked_login

asedb_has_mixed_log_and_data

asedb_runtime_for_open_transactions

asedb_data_cache_hit_ratio

asedb_data_cache_usage

asedb_sql_cache_hit_ratio

asedb_cache_usage

asedb_run_queue_length

asedb_number_of_rollbacks

asedb_number_of_commits

asedb_number_of_transactions

asedb_outstanding_disk_io

asedb_percent_io_busy

asedb_percent_system_busy

asedb_percent_locks_active

asedb_scheduled_jobs_failed_percent

asedb_user_connections_percent

asedb_query_logical_reads

asedb_query_physical_reads

asedb_query_cpu_time

asedb_query_memory_usage

SAP ASE High Availability on Amazon EC2

CloudWatch Application Insights supports the following metrics:

asedb_ha_replication_state

asedb_ha_replication_mode

asedb_ha_replication_latency_in_minutes

SAP NetWeaver

CloudWatch Application Insights supports the following metrics:

Metric	Description
sap_alerts_ResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>ResponseTime.
sap_alerts_ResponseTimeDialog	The SAP response time dialog alert from CCMS (RZ20)>R3Services>Dialog> ResponseTimeDialog.
sap_alerts_ResponseTimeDialogRFC	The SAP response time alert from CCMS (RZ20)>R3Services> Dialog>ResponseTimeDialogRFC.
sap_alerts_DBRequestTime	The SAP response time alert from CCMS (RZ20)>R3Services>Dialog>DBRequestTime.
sap_alerts_FrontendResponseTime	The SAP response time alert from CCMS (RZ20)>R3Services > Dialog>FrontEndResponseTime.
sap_alerts_Database	The SAP system has logged database-related errors. Alert from SM21 or CCMS (RZ20)>R3Syslog>Database.
sap_alerts_QueueTime	The SAP queue time alert from CCMS (RZ20)>R3Services>Dialog>QueueTime.
sap_alerts_AbortedJobs	Failed background jobs in SAP system. Alert from (RZ20)>R3Services > Background>AbortedJobs.

Metric	Description
sap_alerts_BasisSystem	SAP system logged system-level errors. Alert from SM21 or CCMS (RZ20)>R3Syslog>BasisSystem.
sap_alerts_Security	The SAP system logged security-related messages. Alert from SM21 or CCMS (RZ20)>R3Syslog>Security.
sap_alerts_System	The SAP system logged security or audit-related messages. Alert from SM21 or CCMS (RZ20)>Security>System.
sap_alerts_LongRunners	There are long running programs in your SAP system. Alert from CCMS (RZ20)>R3Services >Dialog>LongRunners.
sap_alerts_SqlError	There are SAP database client layer error logs. Alert from CCMS(RZ20)>DatabaseClient>AbapSql>SqlError.
sap_alerts_State	State alert from CCMS (RZ20)>OS Collector >State.
sap_alerts_Shortdumps	Shortdumps alert from ST22 and CCMS (RZ20)>R3Abap>Shortdumps.
sap_alerts_Availability	Availability alert for SAP application server instance from SM21, SM50, SM51, SM66, and CCMS (RZ20)>InstanceAsTask>Availability.
sap_dispatcher_queue_high	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue high count.
sap_dispatcher_queue_max	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue max count.

Metric	Description
sap_dispatcher_queue_now	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue now count.
sap_dispatcher_queue_reads	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue reads count.
sap_dispatcher_queue_writes	The SAPControl Web Service function `GetQueueStatistic` provides the dispatcher queue writes count.
sap_enqueue_server_arguments_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments high.
sap_enqueue_server_arguments_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments max.
sap_enqueue_server_arguments_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments now.
sap_enqueue_server_arguments_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue arguments state.
sap_enqueue_server_backup_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue backup requests.
sap_enqueue_server_cleanup_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue cleanup requests.

Metric	Description
sap_enqueue_server_dequeue_all_requests	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue all requests.
sap_enqueue_server_dequeue_errors	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue errors.
sap_enqueue_server_dequeue_requests	The SAPControl Web Service function `EnqGetStatistic` provides the dequeue requests.
sap_enqueue_server_enqueue_errors	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue errors.
sap_enqueue_server_enqueue_rejects	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue rejects.
sap_enqueue_server_enqueue_requests	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue requests.
sap_enqueue_server_lock_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue lock time.
sap_enqueue_server_lock_wait_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue lock wait time.
sap_enqueue_server_locks_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks high.

Metric	Description
sap_enqueue_server_locks_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks max.
sap_enqueue_server_locks_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks now.
sap_enqueue_server_locks_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue locks state.
sap_enqueue_server_owner_high	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner high.
sap_enqueue_server_owner_max	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner max.
sap_enqueue_server_owner_now	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner now.
sap_enqueue_server_owner_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue owner state.
sap_enqueue_server_replication_state	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue replication state status.
sap_enqueue_server_reporting_requests	The SAPControl Web Service function `EnqGetStatistic` provides the reqporting requests status.

Metric	Description
sap_enqueue_server_server_time	The SAPControl Web Service function `EnqGetStatistic` provides the enqueue server time.
sap_HA_check_failover_config_state	The SAPControl Web Service function `HACheckFailoverConfig` provides the SAP High Availability status.
sap_HA_get_failover_config_HAActive	The SAPControl Web Service function `HAGetFailoverConfig` provides the SAP High Availability Cluster configuration and status.
sap_start_service_processes	The SAPControl Web Service function `GetProcessList` provides the disp+work , IGS, gwrd, icman, message server, and enqueue server processes status.

HA Cluster

CloudWatch Application Insights supports the following metrics:

ha_cluster_pacemaker_stonith_enabled

ha_cluster_corosync_quorate

hanadb_webdispatcher_service_started_status

ha_cluster_pacemaker_nodes

ha_cluster_corosync_ring_errors

ha_cluster_pacemaker_fail_count

Java

CloudWatch Application Insights supports the following metrics:

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_threading_daemonthreadcount

java_lang_classloading_loadedclasscount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon Elastic Container Service (Amazon ECS)

CloudWatch Application Insights supports the following metrics:

Metrics

- CloudWatch built-in metrics

- Container Insights metrics
- Container Insights Prometheus metrics

CloudWatch built-in metrics

CPUReservation

CPUUtilization

MemoryReservation

MemoryUtilization

GPUReservation

Container Insights metrics

ContainerInstanceCount

CpuUtilized

CpuReserved

DeploymentCount

DesiredTaskCount

MemoryUtilized

MemoryReserved

NetworkRxBytes

NetworkTxBytes

PendingTaskCount

RunningTaskCount

ServiceCount

StorageReadBytes

StorageWriteBytes

TaskCount

TaskSetCount

instance_cpu_limit

instance_cpu_reserved_capacity

instance_cpu_usage_total

instance_cpu_utilization

instance_filesystem_utilization

instance_memory_limit

instance_memory_reserved_capacity

instance_memory_utilization

instance_memory_working_set

instance_network_total_bytes

instance_number_of_running_tasks

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Kubernetes on AWS

CloudWatch Application Insights supports the following metrics:

Metrics

- Container Insights metrics
- Container Insights Prometheus metrics

Container Insights metrics

cluster_failed_node_count

cluster_node_count

namespace_number_of_running_pods

node_cpu_limit

node_cpu_reserved_capacity

node_cpu_usage_total

node_cpu_utilization

node_filesystem_utilization

node_memory_limit

node_memory_reserved_capacity

node_memory_utilization

node_memory_working_set

node_network_total_bytes

node_number_of_running_containers

node_number_of_running_pods

pod_cpu_reserved_capacity

pod_cpu_utilization

pod_cpu_utilization_over_pod_limit

pod_memory_reserved_capacity

pod_memory_utilization

pod_memory_utilization_over_pod_limit

pod_network_rx_bytes

pod_network_tx_bytes

service_number_of_running_pods

Container Insights Prometheus metrics

Java JMX metrics

java_lang_memory_heapmemoryusage_used

java_lang_memory_heapmemoryusage_committed

java_lang_operatingsystem_openfiledescriptorcount

java_lang_operatingsystem_maxfiledescriptorcount

java_lang_operatingsystem_freephysicalmemorysize

java_lang_operatingsystem_freeswapspacesize

java_lang_threading_threadcount

java_lang_classloading_loadedclasscount

java_lang_threading_daemonthreadcount

java_lang_garbagecollector_collectiontime_copy

java_lang_garbagecollector_collectiontime_ps_scavenge

java_lang_garbagecollector_collectiontime_parnew

java_lang_garbagecollector_collectiontime_marksweepcompact

java_lang_garbagecollector_collectiontime_ps_marksweep

java_lang_garbagecollector_collectiontime_concurrentmarksweep

java_lang_garbagecollector_collectiontime_g1_young_generation

java_lang_garbagecollector_collectiontime_g1_old_generation

java_lang_garbagecollector_collectiontime_g1_mixed_generation

java_lang_operatingsystem_committedvirtualmemorysize

Amazon FSx

CloudWatch Application Insights supports the following metrics:

DataReadBytes

DataWriteBytes

DataReadOperations

DataWriteOperations

MetadataOperations

FreeStorageCapacity

FreeDataStorageCapacity

LogicalDiskUsage

PhysicalDiskUsage

Amazon VPC

CloudWatch Application Insights supports the following metrics:

NetworkAddressUsage

NetworkAddressUsagePeered

VPCFirewallQueryVolume

Amazon VPC NAT gateways

CloudWatch Application Insights supports the following metrics:

ErrorPortAllocation

IdleTimeoutCount

Amazon Route 53 health check

CloudWatch Application Insights supports the following metrics:

ChildHealthCheckHealthyCount

ConnectionTime

HealthCheckPercentageHealthy

HealthCheckStatus

SSLHandshakeTime

TimeToFirstByte

Amazon Route 53 hosted zone

CloudWatch Application Insights supports the following metrics:

DNSQueries

DNSSECInternalFailure

DNSSECKeySigningKeysNeedingAction

DNSSECKeySigningKeyMaxNeedingActionAge

DNSSECKeySigningKeyAge

Amazon Route 53 Resolver endpoint

CloudWatch Application Insights supports the following metrics:

EndpointHealthyENICount

EndpointUnHealthyENICount

InboundQueryVolume

OutboundQueryVolume

OutboundQueryAggregateVolume

AWS Network Firewall rule group

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupQueryVolume

AWS Network Firewall rule group association

CloudWatch Application Insights supports the following metrics:

FirewallRuleGroupVpcQueryVolume

Metrics with data points requirements

For metrics without an obvious default threshold to alarm on, Application Insights waits until the metric has enough data points to predict a reasonable threshold to alarm on. The metric data points requirement that CloudWatch Application Insights checks before an alarm is created are:

- The metric has at least 100 data points from the past 15 to the past 2 days.
- The metric has at least 100 data points from the last day.

The following metrics follow these data points requirements. Note that CloudWatch agent metrics require up to one hour to create alarms.

Metrics

- AWS/ApplicationELB
- AWS/AutoScaling
- AWS/EC2
- Elastic Block Store (EBS)
- AWS/ELB
- AWS/RDS
- AWS/Lambda
- AWS/SQS
- AWS/CWAgent
- AWS/DynamoDB
- AWS/S3
- AWS/States
- AWS/ApiGateway
- AWS/SNS

AWS/ApplicationELB

ActiveConnectionCount

ConsumedLCUs

HTTPCode_ELB_4XX_Count

HTTPCode_Target_2XX_Count

HTTPCode_Target_3XX_Count

HTTPCode_Target_4XX_Count

HTTPCode_Target_5XX_Count

NewConnectionCount

ProcessedBytes

TargetResponseTime

UnHealthyHostCount

AWS/AutoScaling

GroupDesiredCapacity

GroupInServiceInstances

GroupMaxSize

GroupMinSize

GroupPendingInstances

GroupStandbyInstances

GroupTerminatingInstances

GroupTotalInstances

AWS/EC2

CPUCreditBalance

CPUCreditUsage

CPUSurplusCreditBalance

CPUSurplusCreditsCharged

CPUUtilization

DiskReadBytes

DiskReadOps

DiskWriteBytes

DiskWriteOps

EBSByteBalance%

EBSIOBalance%

EBSReadBytes

EBSReadOps

EBSWriteBytes

EBSWriteOps

NetworkIn

NetworkOut

NetworkPacketsIn

NetworkPacketsOut

Elastic Block Store (EBS)

VolumeReadBytes

VolumeWriteBytes

VolumeReadOps

VolumeWriteOps

VolumeTotalReadTime

VolumeTotalWriteTime

VolumeIdleTime

VolumeQueueLength

VolumeThroughputPercentage

VolumeConsumedReadWriteOps

BurstBalance

AWS/ELB

EstimatedALBActiveConnectionCount

EstimatedALBConsumedLCUs

EstimatedALBNewConnectionCount

EstimatedProcessedBytes

HTTPCode_Backend_4XX

HTTPCode_Backend_5XX

HealthyHostCount

Latency

RequestCount

SurgeQueueLength

UnHealthyHostCount

AWS/RDS

ActiveTransactions

AuroraBinlogReplicaLag

AuroraReplicaLag

BackupRetentionPeriodStorageUsed

BinLogDiskUsage

BlockedTransactions

CPUCreditBalance

CommitLatency

CommitThroughput

DDLLatency

DDLThroughput

DMLLatency

DMLThroughput

DatabaseConnections

Deadlocks

DeleteLatency

DeleteThroughput

DiskQueueDepth

EngineUptime

FreeLocalStorage

FreeStorageSpace

FreeableMemory

InsertLatency

InsertThroughput

LoginFailures

NetworkReceiveThroughput

NetworkThroughput

NetworkTransmitThroughput

Queries

ReadIOPS

ReadThroughput

SelectLatency

SelectThroughput

SnapshotStorageUsed

TotalBackupStorageBilled

UpdateLatency

UpdateThroughput

VolumeBytesUsed

VolumeReadIOPs

VolumeWriteIOPs

WriteIOPS

WriteThroughput

AWS/Lambda

Errors

DeadLetterErrors

Duration

Throttles

IteratorAge

ProvisionedConcurrencySpilloverInvocations

AWS/SQS

ApproximateAgeOfOldestMessage

ApproximateNumberOfMessagesDelayed

ApproximateNumberOfMessagesNotVisible

ApproximateNumberOfMessagesVisible

NumberOfEmptyReceives

NumberOfMessagesDeleted

NumberOfMessagesReceived

NumberOfMessagesSent

AWS/CWAgent

LogicalDisk % Free Space

Memory % Committed Bytes In Use

Memory Available Mbytes

Network Interface Bytes Total/sec

Paging File % Usage

PhysicalDisk % Disk Time

PhysicalDisk Avg. Disk sec/Read

PhysicalDisk Avg. Disk sec/Write

PhysicalDisk Disk Read Bytes/sec

PhysicalDisk Disk Reads/sec

PhysicalDisk Disk Write Bytes/sec

PhysicalDisk Disk Writes/sec

Processor % Idle Time

Processor % Interrupt Time

Processor % Processor Time

Processor % User Time

SQLServer:Access Methods Forwarded Records/sec

SQLServer:Access Methods Page Splits/sec

SQLServer:Buffer Manager Buffer cache hit ratio

SQLServer:Buffer Manager Page life expectancy

SQLServer:Database Replica File Bytes Received/sec

SQLServer:Database Replica Log Bytes Received/sec

SQLServer:Database Replica Log remaining for undo

SQLServer:Database Replica Log Send Queue

SQLServer:Database Replica Mirrored Write Transaction/sec

SQLServer:Database Replica Recovery Queue

SQLServer:Database Replica Redo Bytes Remaining

SQLServer:Database Replica Redone Bytes/sec

SQLServer:Database Replica Total Log requiring undo

SQLServer:Database Replica Transaction Delay

SQLServer:General Statistics Processes blocked

SQLServer:SQL Statistics Batch Requests/sec

SQLServer:SQL Statistics SQL Compilations/sec

SQLServer:SQL Statistics SQL Re-Compilations/sec

System Processor Queue Length

TCPv4 Connections Established

TCPv6 Connections Established

AWS/DynamoDB

ConsumedReadCapacityUnits

ConsumedWriteCapacityUnits

ReadThrottleEvents

WriteThrottleEvents

TimeToLiveDeletedItemCount

ConditionalCheckFailedRequests

TransactionConflict

ReturnedRecordsCount

PendingReplicationCount

ReplicationLatency

AWS/S3

ReplicationLatency

BytesPendingReplication

OperationsPendingReplication

4xxErrors

5xxErrors

AllRequests

GetRequests

PutRequests

DeleteRequests

HeadRequests

PostRequests

SelectRequests

ListRequests

SelectBytesScanned

SelectBytesReturned

FirstByteLatency

TotalRequestLatency

BytesDownloaded

BytesUploaded

AWS/States

ActivitiesScheduled

ActivitiesStarted

ActivitiesSucceeded

ActivityScheduleTime

ActivityRuntime

ActivityTime

LambdaFunctionsScheduled

LambdaFunctionsStarted

LambdaFunctionsSucceeded

LambdaFunctionScheduleTime

LambdaFunctionRuntime

LambdaFunctionTime

ServiceIntegrationsScheduled

ServiceIntegrationsStarted

ServiceIntegrationsSucceeded

ServiceIntegrationScheduleTime

ServiceIntegrationRuntime

ServiceIntegrationTime

ProvisionedRefillRate

ProvisionedBucketSize

ConsumedCapacity

ThrottledEvents

AWS/ApiGateway

4XXError

IntegrationLatency

Latency

DataProcessed

CacheHitCount

CacheMissCount

AWS/SNS

NumberOfNotificationsDelivered

NumberOfMessagesPublished

NumberOfNotificationsFailed

NumberOfNotificationsFilteredOut

NumberOfNotificationsFilteredOut-InvalidAttributes

NumberOfNotificationsFilteredOut-NoMessageAttributes

NumberOfNotificationsRedrivenToDlq

NumberOfNotificationsFailedToRedriveToDlq

SMSSuccessRate

Recommended metrics

The following table lists the recommended metrics for each component type.

Component type	Workload type	Recommended metric
EC2 instance (Windows servers)	Default/Custom	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Active Directory	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes Database ==> Instances Database Cache % Hit DirectoryServices DRA Pending Replication Operations DirectoryServices DRA Pending Replication Synchronizations

Component type	Workload type	Recommended metric
		DNS Recursive Query Failure/sec LogicalDisk Avg. Disk Queue Length
	Java Application	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
	Microsoft IIS/.NET Web Front-End	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC ASP.NET Applications Requests in Application Queue ASP.NET Requests Queued ASP.NET Application Restarts

Component type	Workload type	Recommended metric
	Microsoft SQL Server Database Tier	CPUUtilization
		StatusCheckFailed
		Processor % Processor Time
		Memory % Committed Bytes In Use
		Memory Available Mbytes
		Paging File % Usage
		System Processor Queue Length
		Network Interface Bytes Total/Sec
		PhysicalDisk % Disk Time
		SQLServer:Buffer Manager Buffer Cache Hit ratio
		SQLServer:Buffer Manager Page Life Expectancy
		SQLServer:General Statistics Processes Blocked
		SQLServer:General Statistics User Connections
		SQLServer:Locks Number of Deadlocks/Sec
		SQLServer:SQL Statistics Batch Requests/Sec

Component type	Workload type	Recommended metric
	MySQL	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	.NET workerpool/Mid-Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes .NET CLR Exceptions # of Exceps Thrown/Sec .NET CLR Memory # Total Committed Bytes .NET CLR Memory % Time in GC

Component type	Workload type	Recommended metric
	.NET Core Tier	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use Memory Available Mbytes
	Oracle	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes
	Postgres	CPUUtilization StatusCheckFailed Processor % Processor Time Memory % Committed Bytes In Use LogicalDisk % Free Space Memory Available Mbytes

Component type	Workload type	Recommended metric
	SharePoint	CPUUtilization
		StatusCheckFailed
		Processor % Processor Time
		Memory % Committed Bytes In Use
		Memory Available Mbytes
		ASP.NET Applications Cache API trims
		ASP.NET Requests Rejected
		ASP.NET Worker Process Restarts
		Memory Pages/sec
		SharePoint Publishing Cache Publishing cache flushes / second
		SharePoint Foundation Executing Time/Page Request
		SharePoint Disk-Based Cache Total number of cache compactions
		SharePoint Disk-Based Cache Blob cache hit ratio
		SharePoint Disk-Based Cache Blob Cache fill ratio

Component type	Workload type	Recommended metric
		SharePoint Disk-Based Cache Blob cache flushes / second ASP.NET Requests Queued ASP.NET Applications Requests in Application Queue ASP.NET Application Restarts LogicalDisk Avg. Disk sec/ Write LogicalDisk Avg. Disk sec/ Read Processor % Interrupt Time
EC2 instance (Linux servers)	Default/Custom	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
	.NET Core Tier or SQL Server Database Tier	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
	Oracle	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent
	Postgres	CPUUtilization StatusCheckFailed disk_used_percent mem_used_percent

Component type	Workload type	Recommended metric
EC2 instance group	SAP HANA multi-node or single node	• hanadb_server_startup_time_variations_seconds • hanadb_level_5_alerts_count • hanadb_level_4_alerts_count • hanadb_out_of_memory_events_count • hanadb_max_trigger_read_ratio_percent • hanadb_max_trigger_write_ratio_percent • hanadb_log_switch_race_ratio_percent • hanadb_time_since_last_savepoint_seconds • hanadb_disk_usage_highlevel_percent • hanadb_current_allocation_limit_used_percent • hanadb_table_allocation_limit_used_percent • hanadb_cpu_usage_percent • hanadb_plan_cache_hit_ratio_percent • hanadb_last_data_backup_age_days

Component type	Workload type	Recommended metric
EBS volume	Any	VolumeReadBytes VolumeWriteBytes VolumeReadOps VolumeWriteOps VolumeQueueLength VolumeThroughputPercentage VolumeConsumedReadWriteOps BurstBalance
Classic ELB	Any	HTTPCode_Backend_4XX HTTPCode_Backend_5XX Latency SurgeQueueLength UnHealthyHostCount
Application ELB	Any	HTTPCode_Target_4XX_Count HTTPCode_Target_5XX_Count TargetResponseTime UnHealthyHostCount

Component type	Workload type	Recommended metric
RDS Database instance	Any	CPUUtilization ReadLatency WriteLatency BurstBalance FailedSQLServerAgentJobsCount
RDS Database cluster	Any	CPUUtilization CommitLatency DatabaseConnections Deadlocks FreeableMemory NetworkThroughput VolumeBytesUsed
Lambda Function	Any	Duration Errors IteratorAge ProvisionedConcurrencySpilloverInvocations Throttles

Component type	Workload type	Recommended metric
SQS Queue	Any	ApproximateAgeOfOldestMessage ApproximateNumberOfMessagesVisible NumberOfMessagesSent
Amazon DynamoDB table	Any	SystemErrors UserErrors ConsumedReadCapacityUnits ConsumedWriteCapacityUnits ReadThrottleEvents WriteThrottleEvents ConditionalCheckFailedRequests TransactionConflict

Component type	Workload type	Recommended metric
Amazon S3 bucket	Any	If replication configuration with Replication Time Control (RTC) is enabled: ReplicationLatency BytesPendingReplication OperationsPendingReplication If request metrics are turned on: 5xxErrors 4xxErrors BytesDownloaded BytesUploaded

Component type	Workload type	Recommended metric
AWS Step Functions	Any	**General** • ExecutionThrottled • ExecutionsAborted • ProvisionedBucketSize • ProvisionedRefillRate • ConsumedCapacity **If state machine type is EXPRESS or log group level is OFF** • ExecutionsFailed • ExecutionsTimedOut **If state machine has Lambda functions** • LambdaFunctionsFailed • LambdaFunctionsTimedOut **If state machine has activities** • ActivitiesFailed • ActivitiesTimedOut • ActivitiesHeartbeatTimedOut **If state machine has service integrations** • ServiceIntegrationsFailed

Component type	Workload type	Recommended metric
		• ServiceIntegrationsTimedOut
API Gateway REST API stage	Any	• 4XXErrors • 5XXErrors • Latency

Component type	Workload type	Recommended metric
ECS Cluster	Any	CpuUtilized
		MemoryUtilized
		NetworkRxBytes
		NetworkTxBytes
		RunningTaskCount
		PendingTaskCount
		StorageReadBytes
		StorageWriteBytes
		CPUReservation (EC2 Launch Type only)
		CPUUtilization (EC2 Launch Type only)
		MemoryReservation (EC2 Launch Type only)
		MemoryUtilization (EC2 Launch Type only)
		GPUReservation (EC2 Launch Type only)
		instance_cpu_utilization (EC2 Launch Type only)
		instance_filesystem_utilization (EC2 Launch Type only)
		instance_memory_utilization (EC2 Launch Type only)

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only)

Component type	Workload type	Recommended metric
	Java Application	CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes CPUReservation (EC2 Launch Type only) CPUUtilization (EC2 Launch Type only) MemoryReservation (EC2 Launch Type only) MemoryUtilization (EC2 Launch Type only) GPUReservation (EC2 Launch Type only) instance_cpu_utilization (EC2 Launch Type only) instance_filesystem_utilization (EC2 Launch Type only) instance_memory_utilization (EC2 Launch Type only)

Component type	Workload type	Recommended metric
		instance_network_total_bytes (EC2 Launch Type only) java_lang_threading_threadcount java_lang_classloading_loadedclasscount java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize
ECS Service	Any	CPUUtilization MemoryUtilization CpuUtilized MemoryUtilized NetworkRxBytes NetworkTxBytes RunningTaskCount PendingTaskCount StorageReadBytes StorageWriteBytes

Component type	Workload type	Recommended metric
	Java Application	CPUUtilization
		MemoryUtilization
		CpuUtilized
		MemoryUtilized
		NetworkRxBytes
		NetworkTxBytes
		RunningTaskCount
		PendingTaskCount
		StorageReadBytes
		StorageWriteBytes
		java_lang_threading_threadcount
		java_lang_classloading_loadedclasscount
		java_lang_memory_heapmemoryusage_used
		java_lang_memory_heapmemoryusage_committed
		java_lang_operatingsystem_freephysicalmemorysize
		java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
EKS Cluster	Any	cluster_failed_node_count
		node_cpu_reserved_capacity
		node_cpu_utilization
		node_filesystem_utilization
		node_memory_reserved_capacity
		node_memory_utilization
		node_network_total_bytes
		pod_cpu_reserved_capacity
		pod_cpu_utilization
		pod_cpu_utilization_over_pod_limit
		pod_memory_reserved_capacity
		pod_memory_utilization
		pod_memory_utilization_over_pod_limit
		pod_network_rx_bytes
		pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

Component type	Workload type	Recommended metric
Kubernetes Cluster on EC2	Any	cluster_failed_node_count
		node_cpu_reserved_capacity
		node_cpu_utilization
		node_filesystem_utilization
		node_memory_reserved_capacity
		node_memory_utilization
		node_network_total_bytes
		pod_cpu_reserved_capacity
		pod_cpu_utilization
		pod_cpu_utilization_over_pod_limit
		pod_memory_reserved_capacity
		pod_memory_utilization
		pod_memory_utilization_over_pod_limit
		pod_network_rx_bytes
		pod_network_tx_bytes

Component type	Workload type	Recommended metric
	Java Application	cluster_failed_node_count node_cpu_reserved_capacity node_cpu_utilization node_filesystem_utilization node_memory_reserved_capacity node_memory_utilization node_network_total_bytes pod_cpu_reserved_capacity pod_cpu_utilization pod_cpu_utilization_over_pod_limit pod_memory_reserved_capacity pod_memory_utilization pod_memory_utilization_over_pod_limit pod_network_rx_bytes pod_network_tx_bytes java_lang_threading_threadcount java_lang_classloading_loadedclasscount

Component type	Workload type	Recommended metric
		java_lang_memory_heapmemoryusage_used java_lang_memory_heapmemoryusage_committed java_lang_operatingsystem_freephysicalmemorysize java_lang_operatingsystem_freeswapspacesize

The following table lists the recommended processes and process metrics for each component type. CloudWatch Application Insights does not recommend process monitoring for processes that do not run on an instance.

Component type	Workload type	Recommended process	Recommended metric

EC2 instance (Windows servers)	Microsoft IIS/.NET Web Front-End	`w3wp`	`procstat cpu_usage ,`
			
			`procstat memory_rss ,`
			
			`procstat memory_vms ,`
			
			`procstat read_bytes ,`
			
			`procstat write_bytes`

	Microsoft SQL Server Database Tier	`SQLAgent`	`procstat cpu_usage ,`

Component type	Workload type	Recommended process	Recommended metric
			procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlservr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		sqlwriter	procstat cpu_usage , procstat memory_rss

Component type	Workload type	Recommended process	Recommended metric
		Reporting ServicesS ervice	procstat cpu_usage , procstat memory_rss
		MsDtsServr	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
		Msmdsrv	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Component type	Workload type	Recommended process	Recommended metric
	.NET workerpool/ Mid-Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes
	.NET Core Tier	w3wp	procstat cpu_usage , procstat memory_rss , procstat memory_vms , procstat read_bytes , procstat write_bytes

Performance Counter metrics

Performance Counter metrics are recommended for instances only when the corresponding Performance Counter sets are installed on the Windows instances.

Performance Counter metric name	Performance Counter set name
.NET CLR Exceptions # of Exceps Thrown	.NET CLR Exceptions
.NET CLR Exceptions # of Exceps Thrown/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Filters/Sec	.NET CLR Exceptions
.NET CLR Exceptions # of Finallys/Sec	.NET CLR Exceptions
.NET CLR Exceptions Throw to Catch Depth/ Sec	.NET CLR Exceptions
.NET CLR Interop # of CCWs	.NET CLR Interop
.NET CLR Interop # of Stubs	.NET CLR Interop
.NET CLR Interop # of TLB exports/Sec	.NET CLR Interop
.NET CLR Interop # of TLB imports/Sec	.NET CLR Interop
.NET CLR Interop # of Marshaling	.NET CLR Interop
.NET CLR Jit % Time in Jit	.NET CLR Jit
.NET CLR Jit Standard Jit Failures	.NET CLR Jit
.NET CLR Loading % Time Loading	.NET CLR Loading
.NET CLR Loading Rate of Load Failures	.NET CLR Loading
.NET CLR LocksAndThreads Contention Rate/ Sec	.NET CLR LocksAndThreads
.NET CLR LocksAndThreads Queue Length/Sec	.NET CLR LocksAndThreads
.NET CLR Memory # Total Committed Bytes	.NET CLR Memory
.NET CLR Memory % Time in GC	.NET CLR Memory

Performance Counter metric name	Performance Counter set name
.NET CLR Networking 4.0.0.0 HttpWebRequest Average Queue Time	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Aborted/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Failed/Sec	.NET CLR Networking 4.0.0.0
.NET CLR Networking 4.0.0.0 HttpWebRequests Queued/Sec	.NET CLR Networking 4.0.0.0
APP_POOL_WAS Total Worker Process Ping Failures	APP_POOL_WAS
ASP.NET Application Restarts	ASP.NET
ASP.NET Requests Rejected	ASP.NET
ASP.NET Worker Process Restarts	ASP.NET
ASP.NET Applications Cache API trims	ASP.NET Applications
ASP.NET Applications % Managed Processor Time (estimated)	ASP.NET Applications
ASP.NET Applications Errors Total/Sec	ASP.NET Applications
ASP.NET Applications Errors Unhandled During Execution/Sec	ASP.NET Applications
ASP.NET Applications Requests in Application Queue	ASP.NET Applications
ASP.NET Applications Requests/Sec	ASP.NET Applications
ASP.NET Request Wait Time	ASP.NET
ASP.NET Requests Queued	ASP.NET

Performance Counter metric name	Performance Counter set name
Database ==> Instances Database Cache % Hit	Database ==> Instances
Database ==> Instances I/O Database Reads Average Latency	Database ==> Instances
Database ==> Instances I/O Database Reads/sec	Database ==> Instances
Database ==> Instances I/O Log Writes Average Latency	Database ==> Instances
DirectoryServices DRA Pending Replication Operations	DirectoryServices
DirectoryServices DRA Pending Replication Synchronizations	DirectoryServices
DirectoryServices LDAP Bind Time	DirectoryServices
DNS Recursive Queries/sec	DNS
DNS Recursive Query Failure/sec	DNS
DNS TCP Query Received/sec	DNS
DNS Total Query Received/sec	DNS
DNS Total Response Sent/sec	DNS
DNS UDP Query Received/sec	DNS
HTTP Service Request Queues CurrentQueueSize	HTTP Service Request Queues
LogicalDisk % Free Space	LogicalDisk
LogicalDisk Avg. Disk sec/Write	LogicalDisk
LogicalDisk Avg. Disk sec/Read	LogicalDisk

Performance Counter metric name	Performance Counter set name
LogicalDisk Avg. Disk Queue Length	LogicalDisk
Memory % Committed Bytes In Use	Memory
Memory Available Mbytes	Memory
Memory Pages/Sec	Memory
Memory Long-Term Average Standby Cache Lifetime (s)	Memory
Network Interface Bytes Total/Sec	Network Interface
Network Interface Bytes Received/sec	Network Interface
Network Interface Bytes Sent/sec	Network Interface
Network Interface Current Bandwidth	Network Interface
Paging File % Usage	Paging File
PhysicalDisk % Disk Time	PhysicalDisk
PhysicalDisk Avg. Disk Queue Length	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Read	PhysicalDisk
PhysicalDisk Avg. Disk Sec/Write	PhysicalDisk
PhysicalDisk Disk Read Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Reads/Sec	PhysicalDisk
PhysicalDisk Disk Write Bytes/Sec	PhysicalDisk
PhysicalDisk Disk Writes/Sec	PhysicalDisk
Processor % Idle Time	Processor
Processor % Interrupt Time	Processor

Performance Counter metric name	Performance Counter set name
Processor % Processor Time	Processor
Processor % User Time	Processor
SharePoint Disk-Based Cache Blob Cache fill ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache flushes / second	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Blob cache hit ratio	SharePoint Disk-Based Cache
SharePoint Disk-Based Cache Total number of cache compactions	SharePoint Disk-Based Cache
SharePoint Foundation Executing Time/Page Request	SharePoint Foundation
SharePoint Publishing Cache Publishing cache flushes / second	SharePoint Publishing Cache
Security System-Wide Statistics Kerberos Authentications	Security System-Wide Statistics
Security System-Wide Statistics NTLM Authentications	Security System-Wide Statistics
SQLServer:Access Methods Forwarded Records/Sec	SQLServer:Access Methods
SQLServer:Access Methods Full Scans/Sec	SQLServer:Access Methods
SQLServer:Access Methods Page Splits/Sec	SQLServer:Access Methods
SQLServer:Buffer Manager Buffer cache hit Ratio	SQLServer:Buffer Manager

Performance Counter metric name	Performance Counter set name
SQLServer:Buffer Manager Page life Expectanc y	SQLServer:Buffer Manager
SQLServer:Database Replica File Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log Bytes Received/sec	SQLServer:Database Replica
SQLServer:Database Replica Log remaining for undo	SQLServer:Database Replica
SQLServer:Database Replica Log Send Queue	SQLServer:Database Replica
SQLServer:Database Replica Mirrored Write Transaction/sec	SQLServer:Database Replica
SQLServer:Database Replica Recovery Queue	SQLServer:Database Replica
SQLServer:Database Replica Redo Bytes Remaining	SQLServer:Database Replica
SQLServer:Database Replica Redone Bytes/sec	SQLServer:Database Replica
SQLServer:Database Replica Total Log requiring undo	SQLServer:Database Replica
SQLServer:Database Replica Transaction Delay	SQLServer:Database Replica
SQLServer:General Statistics Processes Blocked	SQLServer:General Statistics
SQLServer:General Statistics User Connections	SQLServer:General Statistics
SQLServer:Latches Average Latch Wait Time (ms)	SQLServer:Latches
SQLServer:Locks Average Wait Time (ms)	SQLServer:Locks

Performance Counter metric name	Performance Counter set name
SQLServer:Locks Lock Timeouts/Sec	SQLServer:Locks
SQLServer:Locks Lock Waits/Sec	SQLServer:Locks
SQLServer:Locks Number of Deadlocks/Sec	SQLServer:Locks
SQLServer:Memory Manager Memory Grants Pending	SQLServer:Memory Manager
SQLServer:SQL Statistics Batch Requests/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Compilations/Sec	SQLServer:SQL Statistics
SQLServer:SQL Statistics SQL Re-Compilations/Sec	SQLServer:SQL Statistics
System Processor Queue Length	System
TCPv4 Connections Established	TCPv4
TCPv6 Connections Established	TCPv6
W3SVC_W3WP File Cache Flushes	W3SVC_W3WP
W3SVC_W3WP File Cache Misses	W3SVC_W3WP
W3SVC_W3WP Requests/Sec	W3SVC_W3WP
W3SVC_W3WP URI Cache Flushes	W3SVC_W3WP
W3SVC_W3WP URI Cache Misses	W3SVC_W3WP
Web Service Bytes Received/Sec	Web Service
Web Service Bytes Sent/Sec	Web Service
Web Service Connection Attempts/Sec	Web Service
Web Service Current Connections	Web Service

Performance Counter metric name	Performance Counter set name
Web Service Get Requests/Sec	Web Service
Web Service Post Requests/Sec	Web Service

Using the resource health view in the CloudWatch console

You can use the resource health view to automatically discover, manage, and visualize the health and performance of hosts across their applications in a single view. You can visualize the health of their hosts by a performance dimension such as CPU or memory, and slice and dice hundreds of hosts in a single view using filters. You can filter by tags or by use cases, such as hosts in the same Auto Scaling group or hosts that use the same load balancer,

Prerequisites

To make sure that you get the full benefit of the resource health view, check that you have the following prerequisites.

- To see the memory utilization of your hosts and use it as a filter, you must install the CloudWatch agent on your hosts and set it up to send a memory metric to CloudWatch in the default `CWAgent` namespace. On Linux and macOS instances, the CloudWatch agent must send the `mem_used_percent` metric. On Windows instances, the agent must send the `Memory % Committed Bytes In Use` metric. These metrics are included if you use the wizard to create the CloudWatch agent configuration file and select any of the pre-defined sets of metrics. Metrics collected by the CloudWatch agent are billed as custom metrics. For more information, see Installing the CloudWatch agent.

  When you use the CloudWatch agent to collect these memory metrics to use with the resource health view, you must include the following section in the CloudWatch agent configuration file. This section contains the default dimension settings and is created by default, so do not change any part of this section to anything different than what is shown in the following example.

```
"append_dimensions": {
  "ImageId": "${aws:ImageId}",
  "InstanceId": "${aws:InstanceId}",
  "InstanceType": "${aws:InstanceType}",
  "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
```

```
},
```

- To view all the information available in the resource health view, you must be signed in to an account that has the following permissions. If you are signed on with fewer permissions, you can still use the resource health view but some performance data will not be visible.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Describe*",
                "cloudwatch:Get*",
                "cloudwatch:List*",
                "logs:Get*",
                "logs:Describe*",
                "sns:Get*",
                "sns:List*",
                "ec2:DescribeInstances",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeRegions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

To view resource health in your account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Insights**, **EC2 Resource Health**.

   The resource health page appears, showing a square for each host in your account. Each square is colored based on the current status of that host, based on the setting for **Color by**. Host squares with an alarm symbol have one or more alarms currently in ALARM state.

You can see up to 500 hosts in a single view. If you have more hosts in your account, use the filter settings in step 6 of this procedure.

3. To change what criteria is used to show each host's health, choose a setting for **Color by**. You can choose **CPU Utilization**, **Memory Utilization**, or **Status check**. Memory utilization metrics are available only for hosts that are running the CloudWatch agent and have it configured to collect memory metrics and send them to the default `CWAgent` namespace. For more information, see Collect metrics, logs, and traces using the CloudWatch agent.
4. To change the thresholds and the colors that are used for the health indicators in the grid, choose the gear icon above the grid.
5. To toggle whether to show alarms in the host grid, choose or clear **Show alarms across all metrics**.
6. To split the hosts in the map into groups, choose a grouping criteria for **Group by**.
7. To narrow the view to fewer hosts, choose a filter criteria for **Filter by**. You can filter by tags and by resource groupings such as Auto Scaling group, instance type, security group, and more.
8. To sort hosts, choose a sorting criteria for **Sort by**. You can sort by status check results, instance state, CPU or memory utilization, and the number of alarms that are in ALARM state.
9. To see more information about a host, choose the square that represents that host. A popup pane appears. To then dive deeper into information about that host, choose **View dashboard** or **View on list**.

Application performance monitoring (APM)

CloudWatch Application Signals provides application performance monitoring (APM) features such as pre-built, standardized dashboards for critical application metrics, correlated trace spans, and a application map to enable you to visualize interactions between applications and their dependencies. You can also search and analyze transaction spans and trace summaries to debug distributed application issues in a business context, for cases such as troubleshooting customer support tickets or finding top impacted customers. You can also create Service Level Objectives (SLOs) to closely track the performance KPIs of critical operations in your application, enabling you to easily identify and triage operations that do not meet your business KPIs.

See the following sections for an overview of these troubleshooting capabilities:

- Monitor the operational health of your applications with Application Signals
- Searching and analyzing spans

Start collecting application metrics and traces

Get the most integrated application performance monitoring experience by auto-instrumenting applications to easily collect telemetry, whether they are running in Amazon EKS clusters, Amazon EC2, Amazon ECS, Kubernetes, Lambda, or on-premise. Optionally, you can also use OpenTelemetry with Application Signals to collect telemetry.

> **Note**
>
> You must enable transaction search to get all APM features along with a new unified pricing for CloudWatch Application Signals, inclusive of X-Ray traces and application transaction spans. For more information about pricing, see Amazon CloudWatch Pricing.

Topics

- Application Signals
- Service level objectives (SLOs)
- Transaction Search
- Synthetic monitoring (canaries)

- CloudWatch RUM

Application Signals

CloudWatch Application Signals helps you monitor and improve application performance on AWS. It automatically collects data from your applications running on services like Amazon EC2, Amazon ECS, and Lambda. You can use CloudWatch Application Signals for the following:

- Monitor application health in real time
- Track performance against business goals
- View relationships between services and dependencies
- Quickly identify and resolve performance issues

- Enable Application Signals to automatically collect metrics and traces from your applications, and display key metrics such as call volume, availability, latency, faults, and errors. Quickly see and triage current operational health, and whether your applications are meeting their longer-term performance goals, without writing custom code or creating dashboards.
- Create and monitor service-level objectives (SLOs) with Application Signals. Easily create and track status of SLOs related to CloudWatch metrics, including the new standard application metrics that Application Signals collects. See and track the service level indicator (SLI) status of your application services within a services list and topology map. Create alarms to track your SLOs, and track the new standard application metrics that Application Signals collects.
- See a map of your application topology that Application Signals automatically discovers, that gives you a visual representation of your applications, dependencies, and their connectivity.
- Application Signals works with CloudWatch RUM, CloudWatch Synthetics canaries, AWS Service Catalog AppRegistry, and Amazon EC2 Auto Scaling to display your client pages, Synthetics canaries, and application names within dashboards and maps.

Supported languages and architectures

Application Signals supports Java, Python, Node.js, and .NET applications.

Application Signals is supported and tested on Amazon EKS, Amazon ECS, and Amazon EC2. On Amazon EKS clusters, it automatically discovers the names of your services and clusters. On other architectures, you must supply the names of services and environments when you enable those services for Application Signals.

The instructions for enabling Application Signals on Amazon EC2 should work on any architecture that supports the CloudWatch agent and AWS Distro for OpenTelemetry. However, the instructions have not been tested on architectures other than Amazon ECS and Amazon EC2.

Supported Regions

Application Signals is supported in every commercial Region except for Canada West (Calgary).

Topics

- Features
- Permissions required for Application Signals
- Supported systems
- Supported instrumentation setups
- Enable Application Signals in your account
- (Optional) Try out Application Signals with a sample app
- Enable your applications on Amazon EKS clusters
- Enable your applications on Amazon EC2
- Enable your applications on Amazon ECS
- Enable your applications on Kubernetes
- Enable your applications on Lambda
- Monitor service events
- Debug applications with Dynamic Instrumentation
- Troubleshooting your Application Signals installation
- (Optional) Configuring Application Signals
- Monitor the operational health of your applications with Application Signals
- Metrics collected by Application Signals
- Custom metrics with Application Signals

Features

- **Use Application Signals for daily application monitoring** – Use Application Signals within the CloudWatch console, as part of daily application monitoring:
  1. If you have created service level objectives (SLOs) for your services, start with the Service Level Objectives (SLO) page. This gives you an immediate view of the health of your most

critical services, operations, and dependencies. Choose the service, operation, or dependency name for an SLO to open the Service detail page and see detailed service information as you troubleshoot issues.

2. Open the Services page to see a summary of all your services, and quickly see services with the highest fault rate or latency. If you have created SLOs, look at the Services table to see which services have unhealthy service level indicators (SLIs). If a particular service is in an unhealthy state, select the service to open the Service detail page and see service operations, dependencies, Synthetics canaries, and client requests. Select a point in a graph to see correlated traces so that you can troubleshoot and identify the root cause of operational issues.
3. If new services have been deployed or dependencies have changed, open the Application Map to inspect your application topology. See a map of your applications that shows the relationship between clients, Synthetics canaries, services, and dependencies. Quickly see SLI health, view key metrics such as call volume, fault rate, and latency, and drill down to see more detailed information in the Service detail page.

Using Application Signals incurs charges. For information about CloudWatch pricing, see Amazon CloudWatch Pricing.

> **Note**
>
> It is not necessary to enable Application Signals to use CloudWatch Synthetics, or CloudWatch RUM. However, Synthetics and CloudWatch RUM work with Application Signals to provide benefits when you use these features together.

- **Application Signals cross-account** – With Application Signals cross-account observability, you can monitor and troubleshoot your applications that span multiple AWS accounts within a single Region.

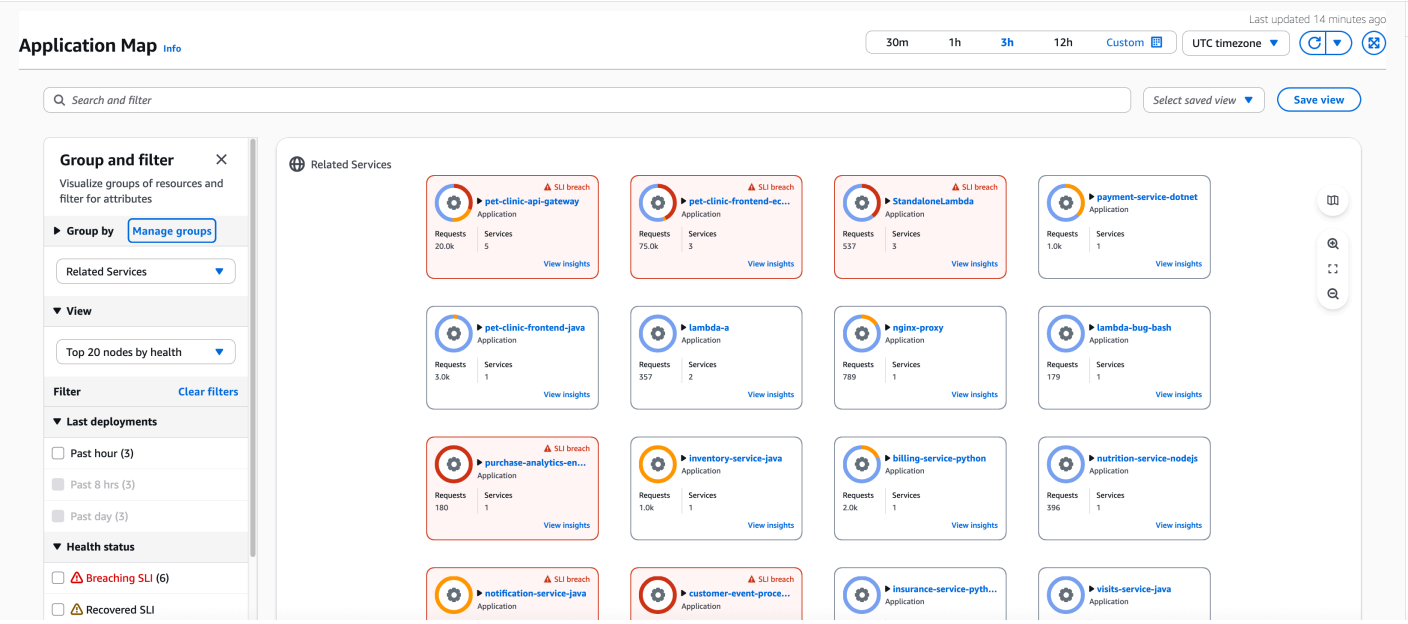
  You can use Amazon CloudWatch Observability Access Manager to set up one or more of your AWS accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a sink in your monitoring account. You use the sink to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

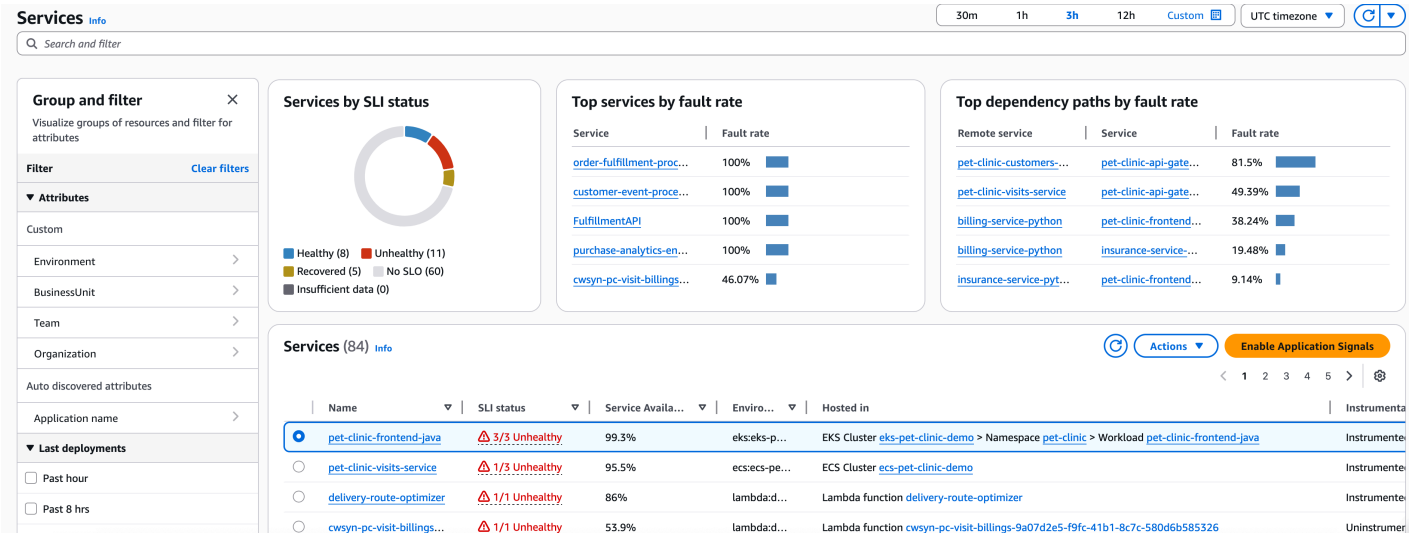
  For proper functionality of Application Signals cross-account observability, make sure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Application Signals services and service level objectives (SLOs)
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs
- Traces in AWS X-Ray

- **Dynamic service grouping and filtering** – Group and filter services with Application Signals' dynamic grouping capabilities. Automatically aggregate metrics and SLIs of services within groups, allowing you to start from a group view and dive deep into specific problematic areas. Application Signals provides two default groupings: "Environment" grouping that organizes by service environment, and "Related services" grouping that groups services based on their dependencies. For example, in Related services grouping, if Service A calls Service B, which calls Service C, they're grouped under Service A. Beyond default groupings, create custom groups by selecting services that align with your organizational needs, such as Business unit or Team.

  Create custom groupings using AWS tags or OpenTelemetry attributes that align with your team structure, business domains, or operational requirements. Custom groupings enable you to organize services according to your specific monitoring and troubleshooting workflows. For more information, see Configuring custom groups.

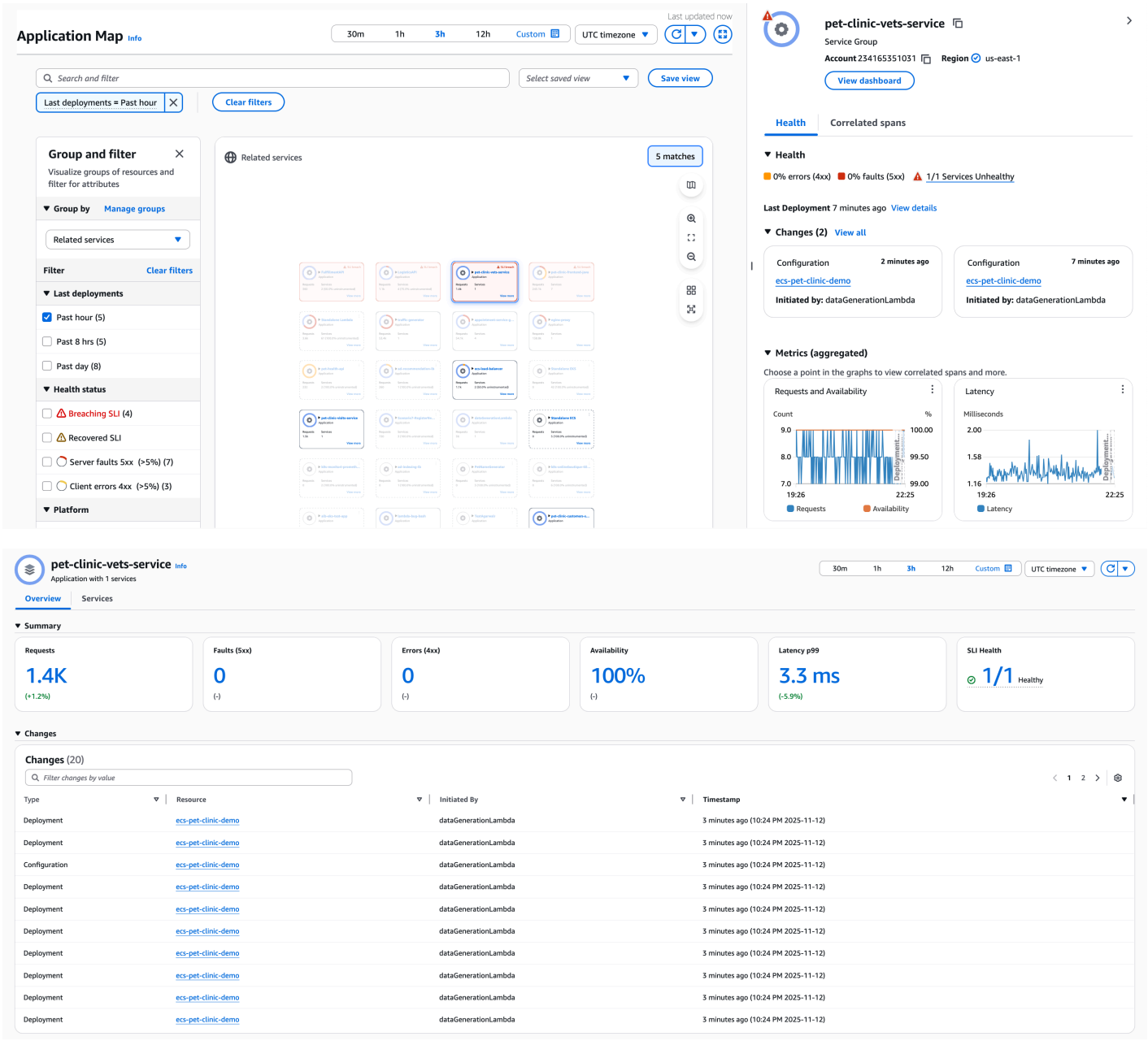




- **Change Events** – Track change events across your application with Application Signals' automatic processing of CloudTrail events. Monitor configuration and deployment events for services and their dependencies, providing immediate context for operational analysis and troubleshooting. Change event detection is enabled alongside service discovery enablement through the CloudWatch Console or StartDiscovery API. For Amazon EKS services, deployment detection requires that the Amazon EKS services are instrumented with the Application Signals instrumentation SDK.

  Change events are supported for the following resources:

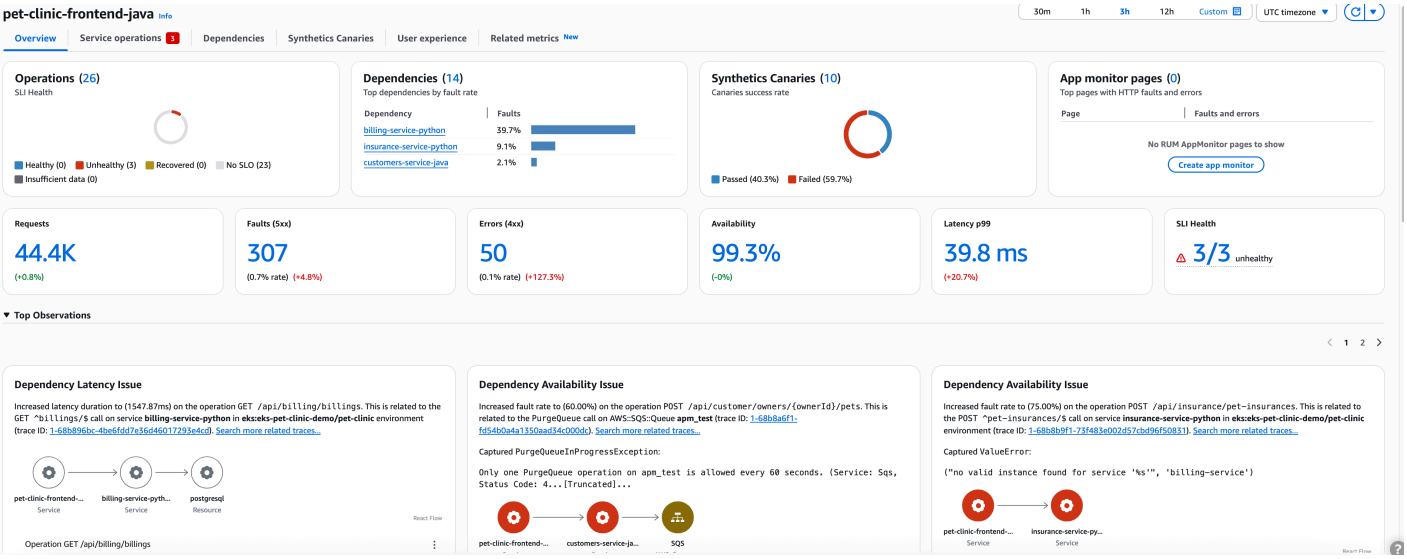
  - Autoscaling Group
  - EKS Cluster
  - EKS Workload (only deployments)
  - ECS Cluster and Service
  - ELB Load balancer and Target Group
  - Lambda Function
  - BedrockAgentCore Runtime and RuntimeEndpoint



- **Automated audit findings** – Discover critical insights through Application Signals' automated audit findings. The service analyzes your applications to report significant observations and potential problems, simplifying root cause analysis. These automated findings consolidate relevant traces, eliminating the need to navigate through multiple clicks. The audit system helps teams quickly identify issues and their underlying causes, enabling faster problem resolution.

  Application Signals employs advanced analytics to detect patterns, highlight resource inefficiencies, and suggest optimization opportunities. Findings are prioritized based on severity and potential business impact, enabling teams to focus on the most critical issues first. Get

actionable recommendations for improving service reliability and performance without manual analysis.



Permissions required for Application Signals

This section explains the permissions necessary for you to enable, manage, and operate Application Signals.

Permissions to enable and manage Application Signals

To manage Application Signals, you must be signed on with the required permissions.
To view the contents of the **CloudWatchApplicationSignalsFullAccess** policy, see CloudWatchApplicationSignalsFullAccess.

To enable Application Signals on Amazon EC2, or custom architectures, see Enable Application Signals on Amazon EC2. To enable and manage Application Signals on Amazon EKS using the Amazon CloudWatch Observability EKS add-on, you need the following permissions.

> **Important**
>
> These permissions include `iam:PassRole` with `Resource "*"` and `eks:CreateAddon` with `Resource "*"`. These are powerful permissions and you should use caution in granting them.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
    "Sid": "CloudWatchApplicationSignalsEksAddonManagementPermissions",
    "Effect": "Allow",
    "Action": [
    "eks:AccessKubernetesApi",
    "eks:CreateAddon",
    "eks:DescribeAddon",
    "eks:DescribeAddonConfiguration",
    "eks:DescribeAddonVersions",
    "eks:DescribeCluster",
    "eks:DescribeUpdate",
    "eks:ListAddons",
    "eks:ListClusters",
    "eks:ListUpdates",
    "iam:ListRoles",
    "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
    "StringEquals": {
    "iam:PassedToService": [
    "eks.amazonaws.com",
    "application-signals.cloudwatch.amazonaws.com"
    ]
    }
    }
    },
    {
    "Sid":
 "CloudWatchApplicationSignalsEksCloudWatchObservabilityAddonManagementPermissions",
    "Effect": "Allow",
    "Action": [
    "eks:DeleteAddon",
    "eks:UpdateAddon"
    ],
    "Resource": "arn:aws:eks:*:*:addon/*/amazon-cloudwatch-observability/*"
    }
    ]
```

```
    }
```

The Application Signals dashboard shows the AWS Service Catalog AppRegistry applications that your SLOs are associated with. To see these applications in the SLO pages, you must have the following permissions:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

Operating Application Signals

Service operators who are using Application Signals to monitor services and SLOs must be signed on to an account with read only permissions. To view the contents of the **CloudWatchApplicationSignalsReadOnlyAccess** policy, see CloudWatchApplicationSignalsReadOnlyAccess.

To see which AWS Service Catalog AppRegistry Applications that your SLOs are associated within the Application Signals dashboard, you must have the following permissions:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsTaggingReadPermissions",
            "Effect": "Allow",
```

```
            "Action": "tag:GetResources",
            "Resource": "*"
        }
    ]
}
```

To check if Application Signals on Amazon EKS using the Amazon CloudWatch Observability EKS add-on is enabled, you need to have the following permissions:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchApplicationSignalsResourceExplorerReadPermissions",
            "Effect": "Allow",
            "Action": [
                "resource-explorer-2:ListIndexes",
                "resource-explorer-2:Search"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchApplicationSignalsResourceExplorerSLRPermissions",
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/resource-
explorer-2.amazonaws.com/AWSServiceRoleForResourceExplorer",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "resource-explorer-2.amazonaws.com"
                    ]
                }
            }
        },
        {
```

```
            "Sid":
 "CloudWatchApplicationSignalsResourceExplorerCreateIndexPermissions",
            "Effect": "Allow",
            "Action": [
                "resource-explorer-2:CreateIndex"
            ],
            "Resource": "arn:aws:resource-explorer-2:*:*:index/*"
        }
    ]
}
```

Supported systems

Application Signals is supported and tested on Amazon EKS, native Kubernetes, Amazon ECS, and Amazon EC2. The instructions for enabling Application Signals on Amazon EC2 should work on any platform that supports the CloudWatch agent and AWS Distro for OpenTelemetry.

Topics

- Java compatibility
- .NET compatibility
- PHP compatibility
- Ruby compatibility
- Python compatibility
- Node.js compatibility
- GoLang compatibility
- Runtime version support matrix
- Known issues

Java compatibility

Application Signals supports Java applications, and supports the same Java libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see Supported libraries, frameworks, application servers, and JVMs.

.NET compatibility

Application Signals supports the same .NET libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see Supported instrumentations.

Application Signals supports .NET applications that are running on x86-64 or ARM64 CPUs, and supports the Linux x64, Linux ARM64, and Microsoft Windows Server 2022 x64.

> **Note**
>
> The AWS Distro for Open Telemetry (ADOT) SDK for .NET does not support AWS SDK for .NET V4. Use AWS SDK .NET V3 for full Application Signals support.

PHP compatibility

Application Signals supports PHP applications with OpenTelemetry Zero Code instrumentation. There is no AWS Distro for Open Telemetry (ADOT) SDK available for this purpose. You should use the standard OpenTelemetry Instrumentation SDK with Transaction Search enabled. To start using zero code instrumentation in PHP, follow these steps from the OpenTelemetry PHP Instrumentation docs, PHP zero-code instrumentation . Automatic instrumentation is available for a number of commonly used PHP libraries. For more information, see OpenTelemetry registry.

Ruby compatibility

Application Signals supports Ruby applications with OpenTelemetry Zero Code instrumentation. There is no AWS Distro for Open Telemetry (ADOT) SDK available for this purpose. You should use the standard OpenTelemetry Instrumentation SDK with Transaction Search enabled. To start using zero code instrumentation in Ruby, follow these steps from the OpenTelemetry Ruby Instrumentation docs, Ruby zero-code instrumentation . For a list of released instrumentation libraries, see Registry.

Python compatibility

Application Signals supports the same libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see **Supported packages** at opentelemetry-python-contrib.

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing `PYTHONPATH` environment variable can sometimes cause the application to fail to start. To resolve this, make sure that you set the `PYTHONPATH` environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

Node.js compatibility

Application Signals supports the same Node.js libraries and frameworks as the AWS Distro for OpenTelemetry does. For more information, see Supported instrumentations.

Known limitations about Node.js with ESM

The AWS Distro for Opentelemetry Node.js supports two module systems: ECMAScript Modules (ESM) and CommonJS (CJS). To enable Application Signals, you recommend that you use the CJS module format because OpenTelemetry JavaScript's support of ESM is experimental and a work in progress. For more details, see ECMAScript Modules versus CommonJS on GitHub.

To determine if your application is using CJS and not ESM, make sure that your application does not fulfill the conditions to enable ESM. For more information about these conditions, see Enabling in the Node.js documentation.

The AWS Distro for Opentelemetry Node.js provides limited support for ESM based on OpenTelemetry JavaScript's experimental support for ESM. This means the following:

- The Node.js version must be 18.19.0 or later.
- The Node.js application that you want to instrument must include `@aws/aws-distro-opentelemetry-node-autoinstrumentation` and `@opentelemetry/instrumentation` as dependencies.
- The Node.js application that you want to instrument must start with the following node option:

```
NODE_OPTIONS=' --import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
```

To enable Application Signals with Node.js ESM module format, we provide the different setup for different platforms:

- **Amazon EKS** – the section called “Setting up a Node.js application with the ESM module format”
- **Amazon ECS with sidecar strategy** – Setting up a Node.js application with the ESM module format
- **Amazon ECS with daemon strategy** – Setting up a Node.js application with the ESM module format
- **Amazon ECS with AWS CDK**
- **Amazon EC2** – Setting up a Node.js application with the ESM module format
- **Kubernetes** – the section called “Setting up a Node.js application with the ESM module format”

GoLang compatibility

Application Signals supports GoLang applications with OpenTelemetry Zero Code instrumentation. There is no AWS Distro for Open Telemetry (ADOT) SDK available for this purpose. You should use the standard OpenTelemetry Instrumentation SDK with Transaction Search enabled. To start using zero code instrumentation in GoLang, follow these steps from the OpenTelemetry GoLang Instrumentation docs, Getting Started with OpenTelemetry Go Automatic Instrumentation .

Implementation considerations GoLang instrumentation

Learn about important implementation details for using GoLang instrumentation. This guidance explains how to implement explicit context propagation in GoLang applications and set up Application Signals. Properly implementing GoLang instrumentation helps you track and analyze your application's performance effectively.

Instrumenting the AWS SDK

The Golang auto-instrumentation library does not support AWS SDK instrumentation out of the box. You must use the `otelaws` library instrumentation along with the auto-instrumentation agent:

1. Install the required dependency:

```
go get go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-sdk-go-v2/
otelaws
```

2. Add the following line to the application:

```
otelaws.AppendMiddlewares(&cfg.APIOptions)
```

3. Create subsequent AWS clients with the previous `aws.Config` object:

```
s3Client := s3.NewFromConfig(cfg)
```

The following example will generate spans for AWS calls and integrates with auto-instrumentation.

```
func handleRequest(ctx context.Context) error {
    cfg, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        return err
    }

    // Add OpenTelemetry instrumentation middleware to the AWS config
    otelaws.AppendMiddlewares(&cfg.APIOptions)

    // Create S3 client with the instrumented config
    s3Client := s3.NewFromConfig(cfg)

    // Now any operations with this client will be traced
    // with the context from the upstream call
    _, err = s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
    return err
}
```

For information on configuring the auto-instrumentation executable, see Configuration methods.

Instrumenting HTTP calls

HTTP calls can split traces when Context isn't passed between requests – HTTP clients must use `NewRequestWithContext()` instead of `NewRequest()` to make sure ensures the downstream service uses the same context. When both services have instrumentation agents, the spans connect with the same trace ID to provide end-to-end visibility.

```
func makeDownstreamCall(ctx context.Context, url string) ([]byte, error) {
    client := &http.Client{}

    // Create request with context from the upstream call
    req, err := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)
    if err != nil {
        return nil, err
    }

    // Execute the request
    resp, err := client.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()
}
```

Instrumenting SQL calls

SQL spans may become disconnected from their parent span, causing client calls to be inferred as server spans. This occurs when SQL calls do not receive the context from their upstream handlers. Standard SQL calls like `Query` and `Exec` use `context.Background()` by default, not the context of the upstream caller. Replace standard SQL calls with their context-aware equivalents:

- Use `QueryContext` instead of `Query`
- Use `ExecContext` instead of `Exec`

These methods Pass the upstream request context to the DB calls, maintaining proper trace continuity.

```
func queryDatabase(ctx context.Context, db *sql.DB, userID string) (*sql.Rows, error) {
    // This breaks the trace context
    // row := db.Query("SELECT name FROM users WHERE id = $1", userID)

    // This passes the context from the upstream call for trace continuity
    rows, err := db.QueryContext(ctx, "SELECT name FROM users WHERE id = $1", userID)

    return rows, error
}
```

> **Note**
>
> The `db.system` attribute is not currently supported for SQL calls. This limitation affects CloudWatch's ability to accurately identify database clients. As a result, dependencies will display **UnknownRemoteService** instead of the name of the DB client making the query.

Resource detectors

Go auto-instrumentation does not currently support configuring resource detectors at runtime. The OpenTelemetry community is working on a feature to configure resource detectors using environment variables. Look for this feature in a future update. In the meantime, you can use the CloudWatch Agent with auto-instrumentation to automatically generate host resource attributes.

Runtime version support matrix

Language	Runtime version
Java	JVM versions 8, 11, 17, 21, 23, and 25
Python	Python versions 3.10, 3.11, 3.12, 3.13, 3.14
.NET	.NET 8, 9, 10 and .NET Framework 4.6.2 and higher
Node.js	Node.js versions 18, 20, 22, and 24
PHP	PHP versions 8.0 and higher
Ruby	CRuby >= 3.1, JRuby >= 9.3.2.0, or TruffleRuby >= 22.1
GoLang	Golang versions 1.18 and higher

Known issues

The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability

EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.6.0-eksbuild.1`. The issue is fixed in Java SDK release `v1.32.6` and the Amazon CloudWatch Observability EKS add-on version `v3.0.0-eksbuild.1`.

If you are impacted, either upgrade the Java SDK version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Supported instrumentation setups

You can enable CloudWatch Application Signals with different instrumentation setups. This topic describes each of the setup methods and recommendations based on the method you choose.

Use AWS Distro for OpenTelemetry with the CloudWatch Agent

The most integrated application performance monitoring(APM) experience in CloudWatch is delivered through the AWS Distro for OpenTelemetry (ADOT) SDKs and are used with the CloudWatch Agent to collect application metrics and traces. This option works best if you want to get started with APM in CloudWatch quickly and also use out-of-the box integrations with features, such as Container Insights and CloudWatch Logs. For more information, see Enable Application Signals on Amazon EKS Clusters and Enable Application Signals on Amazon EC2, Amazon ECS, or Kubernetes.

Use the OpenTelemetry SDK and Collector

This setup works for the following use cases:

1. You instrumented your application or plan with OpenTelemetry SDKs and currently are using OpenTelemetry Collector.
2. You're using languages, such as Erlang and Rust, that aren't supported by AWS Distro for OpenTelemetry (ADOT).

For more information, see OpenTelemetry with CloudWatch.

Use the AWS X-Ray SDK and daemon

This option is best if you instrumented your application using X-Ray SDKs and haven't migrated ADOT SDKs or OpenTelemetry SDKs.

For more information, see Transaction Search.

Feature comparison

Feature	ADOT SDK + CloudWatch Agent	Open Telemetry SDK + OpenTelemetry Collector	X-Ray SDKs
AWS Support	Yes	Only for data sent to AWS	Yes
Nonstandard language support	No	Yes	No
Container Insights integration	Yes	No	No
Out of the box logging with CloudWatch Logs	Yes	No	No
Out of the box runtime metrics	Yes	No	No
Always gets metrics on 100% of traffic	Yes	Only at 100% sampling rate	Only at 100% sampl rate

Enable Application Signals in your account

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. You need to do this only once for your account.

To enable CloudWatch Application Signals, do the following.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Services**.
3. Choose **Start discovering your Services**.
4. Select the check box and choose **Start discovering Services**.

Completing this step for the first time in your account creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role. This role grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

5. Choose **Enable Application Signals**.

(Optional) Try out Application Signals with a sample app

To try out CloudWatch Application Signals on a sample app before you instrument your own applications with it, follow the instructions in this section. These instructions use scripts to help you create an Amazon EKS cluster, install a sample application, and instrument the sample application to work with Application Signals.

The sample application is a Spring "Pet Clinic" application that is composed of four microservices. These services run on Amazon EKS on Amazon EC2 and use Application Signals enablement scripts to enable the cluster with the Java, Python, or .NET auto-instrumentation agent.

Requirements

- Application Signals monitors only Java, Python, or .NET applications.
- You must have the AWS CLI installed on the instance. We recommend AWS CLI version 2, but version 1 should also work. For more information about installing the AWS CLI, see Install or update the latest version of the AWS CLI.
- The scripts in this section are intended to be run in Linux and macOS environments. For Windows instances, we recommend that you use AWS CloudShell to run these scripts. For more information about AWS CloudShell, see What is AWS CloudShell?.

- Install a supported version of `kubectl`. You must use a version of `kubectl` within one minor version difference of your Amazon EKS cluster control plane. For example, a 1.26 `kubectl` client works with Kubernetes 1.25, 1.26, and 1.27 clusters. If you already have an Amazon EKS cluster, you might need to configure AWS credentials for `kubectl`. For more information, see Creating or updating a `kubeconfig` file for an Amazon EKS cluster.
- Install `eksctl`. `eksctl` uses the AWS CLI to interact with AWS, which means it uses the same AWS credentials as the AWS CLI. For more information, see Installing or updating `eksctl`.
- Install `jq`. `jq` is required to run the Application Signals enablement scripts. For more information, see Download jq.

Step 1: Download the scripts

To download the scripts to set up CloudWatch Application Signals with a sample app, you can download and uncompress the zipped GitHub project file to a local drive, or you can clone the GitHub project.

To clone the project, open a terminal window and enter the following Git command in a given working directory.

```
git clone https://github.com/aws-observability/application-signals-demo.git
```

Step 2: Build and deploy the sample application

To build and push the sample application images, follow these instructions.

Step 3: Deploy and enable Application Signals and the sample application

Be sure that you have completed the requirements listed in (Optional) Try out Application Signals with a sample app before you complete the following steps.

To deploy and enable Application Signals and the sample application

1. Enter the following command. Replace *`new-cluster-name`* with the name that you want to use for the new cluster. Replace *`region-name`* with the name of the AWS Region, such as `us-west-1`.

   This command sets up the sample app running in a new Amazon EKS cluster with Application Signals enabled.

```
# this script sets up a new cluster, enables Application Signals, and deploys the
# sample application
cd application-signals-demo/scripts/eks/appsignals/one-step && ./setup.sh new-
cluster-name region-name
```

The setup script takes about 30 minutes to run, and does the following:

- Creates a new Amazon EKS cluster in the specified Region.
- Creates the necessary IAM permissions for Application Signals (`arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess` and `arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy`).
- Enables Application Signals by installing the CloudWatch agent and Auto-instrumenting the sample application for CloudWatch metrics and X-Ray traces.
- Deploys the PetClinic Spring sample application in the same Amazon EKS cluster.
- Creates five CloudWatch Synthetics canaries, named `pc-add-vist`, `pc-create-owners`, `pc-visit-pet`, `pc-visit-vet`, `pc-clinic-traffic`. These canaries will run at a one-minute frequency to generate synthetic traffic for the sample app and demonstrate how Synthetics canaries appear in Application Signals.
- Creates four service level objectives (SLOs) for the PetClinic application with the following names:
  - **Availability for Searching an Owner**
  - **Latency for Searching an Owner**
  - **Availability for Registering an Owner**
  - **Latency for Registering an Owner**
- Creates the required IAM role with a custom trust policy granting Application Signals the following permissions:
  - `cloudwatch:PutMetricData`
  - `cloudwatch:GetMetricData`
  - `xray:GetServiceGraph`
  - `logs:StartQuery`
  - `logs:GetQueryResults`

2. (Optional) If you want to review the source code for the PetClinic sample application, you can find them under the root folder.

```
- application-signals-demo
  - spring-petclinic-admin-server
  - spring-petclinic-api-gateway
  - spring-petclinic-config-server
  - spring-petclinic-customers-service
  - spring-petclinic-discovery-server
  - spring-petclinic-vets-service
  - spring-petclinic-visits-service
```

3. To view the deployed PetClinic sample application, run the following command to find the URL:

```
kubectl get ingress
```

Step 4: Monitor the sample application

After completing the steps in the previous section to create the Amazon EKS cluster and deploy the sample application, you can use Application Signals to monitor the application.

> **Note**
>
> For the Application Signals console to start populating, some traffic must reach the sample application. Part of the previous steps created CloudWatch Synthetics canaries that generate traffic to the sample application.

Service health monitoring

After it is enabled, CloudWatch Application Signals automatically discovers and populates a list of services without requiring any additional setup.

To view the list of discovered services and monitor their health

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Services**.
3. To view a service, its operations, and its dependencies, choose the name of one of the services in the list.

This unified, application-centric view helps provide a full perspective of how users are interacting with your service. This can help you triage issues if performance anomalies occur. For complete details about the **Services** view, see Monitor the operational health of your applications with Application Signals.

4. Choose the **Service Operations** tab to see the standard application metrics for that service's operations. The operations are the API operations that the service calls, for example.

   Then, to view the graphs for a single operation of that service, choose that operation name.
5. Choose the **Dependencies** tab to see the dependencies that your application has, along with the critical application metrics for each dependency. Dependencies include AWS services and third-party services that your application calls.
6. To view correlated traces from the service details page, choose a data point in one of the three graphs above the table. This populates a new pane with filtered traces from the time period. These traces are sorted and filtered based on the graph that you chose. For example, if you chose the **Latency** graph, the traces are sorted by service response time.
7. In the CloudWatch console navigation pane, choose **SLOs**. You see the SLOs that the script created for the sample application. For more information about SLOs, see Service level objectives (SLOs).

(Optional) Step 5: Cleanup

When you're finished testing Application signals, you can use a script provided by Amazon to clean up and delete the artifacts created in your account for the sample application. To perform the cleanup, enter the following command. Replace *new-cluster-name* with the name of the cluster that you created for the sample app, and replace *region*-name with the name of the AWS Region, such as `us-west-1`.

```
cd application-signals-demo/scripts/eks/appsignals/one-step && ./cleanup.sh new-
cluster-name region-name
```

Enable your applications on Amazon EKS clusters

CloudWatch Application Signals is supported for Java, Python, Node.js, and .NET applications. To enable Application Signals for your applications on an existing Amazon EKS cluster, you can use the AWS Management Console, AWS CDK, or CloudWatch Observability add-on Auto monitor advanced configuration.

Topics

- Enable Application Signals on an Amazon EKS cluster using the console
- Enable Application Signals on an Amazon EKS cluster using the CloudWatch Observability add-on advanced configuration
- Enable Application Signals on Amazon EKS using AWS CDK
- Enable Application Signals on Amazon EKS using Model Context Protocol (MCP)

Enable Application Signals on an Amazon EKS cluster using the console

To enable CloudWatch Application Signals on your applications on an existing Amazon EKS cluster, use the instructions in this section.

> **Important**
>
> If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see Supported systems before you enable Application Signals.

To enable Application Signals for your applications on an existing Amazon EKS cluster

> **Note**
>
> If you haven't already enabled Application Signals, follow the instructions in Enable Application Signals in your account and then follow the procedure below.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Application Signals**.
3. For **Specify platform**, choose **EKS**.
4. For **Select an EKS cluster**, select the cluster where you want to enable Application Signals.
5. If this cluster does not already have the Amazon CloudWatch Observability EKS add-on enabled, you are prompted to enable it. If this is the case, do the following:
   a. Choose **Add CloudWatch Observability EKS add-on**. The Amazon EKS console appears.
   b. Select the check box for **Amazon CloudWatch Observability** and choose **Next**.

The CloudWatch Observability EKS add-on enables both Application Signals and CloudWatch Container Insights with enhanced observability for Amazon EKS. For more information about Container Insights, see Container Insights.

c. Select the most recent version of the add-on to install.

d. Select an IAM role to use for the add-on. If you choose **Inherit from node**, attach the correct permissions to the IAM role used by your worker nodes. Replace *my-worker-node-role* with the IAM role used by your Kubernetes worker nodes.

```
aws iam attach-role-policy \
--role-name my-worker-node-role \
--policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--policy-arn arn:aws:iam::aws:policy/AWSXRayWriteOnlyAccess
```

e. If you want to create a service role to use the add-on, see Install the CloudWatch agent with the Amazon CloudWatch Observability EKS add-on or the Helm chart.

f. Choose **Next**, confirm the information on the screen, and choose **Create**.

g. In the next screen, choose **Enable CloudWatch Application Signals** to return to the CloudWatch console and finish the process.

6. There are two options for enabling your applications for Application Signals. For consistency, we recommend that you choose one option per cluster.

- The **Console** option is simpler. Using this method causes your pods to restart immediately.
- The **Annotate Manifest File** method gives you more control of when your pods restart, and can also help you manage your monitoring in a more decentralized way if you don't want to centralize it.

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, skip to Setting up a Node.js application with the ESM module format instead.

Console

The **Console** option uses the advanced configuration of the Amazon CloudWatch Observability EKS add-on to setup Application Signals for your services. For more information about the add-on, see (Optional) Additional configuration.

If you don't see a list of workloads and namespaces, make sure you have the right permissions to view them for this cluster. For more information, see Required permissions.

You can either monitor all service workloads by selecting the **Auto monitor** check box or selectively choose specific workloads and namespaces to monitor.

To monitor all service workload(s) with Auto monitor:

1. Select the **Auto monitor** check box to automatically select all service workload(s) in a cluster.
2. Choose **Auto restart** to restart all workload pods to enable Application Signals immediately with AWS Distro for OpenTelemetry auto-instrumentation (ADOT) SDKs injected into your pods.
3. Choose **Done** . When **Auto restart** is selected, CloudWatch Observability EKS add-on will enable Application Signals immediately. Otherwise, Application Signals will be enabled during the next deployment of each workload.

You can monitor single workloads or entire namespaces.

To monitor a single workload:

1. Select the check box by the workload that you want to monitor.
2. Use the **Select language(s)** dropdown list to select the language of the workload. Select the languages that you want to enable Application Signals for, and then choose the check mark icon (✓) to save this selection.

   For Python applications, make sure your application follows the required prerequisites before continuing. For more information, see Python application doesn't start after Application Signals is enabled.

3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject AWS Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To monitor an entire namespace:

1. Select the check box by the namespace that you want to monitor.
2. Use the **Select language(s)** dropdown list to select the language of the namespace. Select the languages that you want to enable Application Signals for, and then choose the check mark icon (✓) to save this selection. This applies it to all workloads in this namespace, whether they are currently deployed or will be deployed in the future.

   For Python applications, make sure your application follows the required prerequisites before continuing. For more information, see Python application doesn't start after Application Signals is enabled.
3. Choose **Done**. The Amazon CloudWatch Observability EKS add-on will immediately inject AWS Distro for OpenTelemetry autoinstrumentation (ADOT) SDKs into your pods and trigger pod restarts to enable collection of application metrics and traces.

To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

Annotate manifest file

In the CloudWatch console, the **Monitor Services** section explains that you must add an annotation to a manifest YAML in the cluster. Adding this annotation auto-instruments the application to send metrics, traces, and logs to Application Signals.

You have two options for the annotation:

- **Annotate Workload** auto-instruments a single workload in the cluster.
- **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps:

- To annotate a single workload:

1. Choose **Annotate Workload**.
2. Paste one of the following lines into the `PodTemplate` section of the workload manifest file.
   - **For Java workloads:** `annotations: instrumentation.opentelemetry.io/inject-java: "true"`
   - **For Python workloads:** `annotations: instrumentation.opentelemetry.io/inject-python: "true"`

     For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.
   - **For .NET workloads** `annotations: instrumentation.opentelemetry.io/inject-dotnet: "true"`

     > **Note**
     >
     > To enable Application Signals for a .NET workload on Alpine Linux (`linux-musl-x64`) based images, add the following annotation.
     >
     > ```
     > instrumentation.opentelemetry.io/otel-dotnet-auto-runtime: "linux-musl-x64"
     > ```
   - **For Node.js workloads:** `annotations: instrumentation.opentelemetry.io/inject-nodejs: "true"`
3. In your terminal, enter `kubectl apply -f` *`your_deployment_yaml`* to apply the change.

- To annotate all workloads in a namespace:
  1. Choose **Annotate Namespace**.
  2. Paste one of the following lines into the metadata section of the namespace manifest file. If the namespace includes Java, Python, and .NET workloads, paste all of the following lines into the namespace manifest file.
     - **If there are Java workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-java: "true"`
     - **If there are Python workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-python: "true"`

For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.

- **If there are .NET workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-dotnet: "true"`
- **If there are Node.JS workloads in the namespace:** `annotations: instrumentation.opentelemetry.io/inject-nodejs: "true"`

3. In your terminal, enter `kubectl apply -f your_namespace_yaml` to apply the change.
4. In your terminal, enter a command to restart all pods in the namespace. An example command to restart deployment workloads is `kubectl rollout restart deployment -n namespace_name`

7. Choose **View Services when done**. This takes you to the Application Signals Services view, where you can see the data that Application Signals is collecting. It might take a few minutes for data to appear.

   To enable Application Signals in another Amazon EKS cluster, choose **Enable Application Signals** from the **Services** screen.

   For more information about the **Services** view, see Monitor the operational health of your applications with Application Signals.

> **Note**
>
> If you're using a pre-fork server (WSGI or ASGI) such as Gunicorn, uWSGI, or Uvicorn for your Python application, see No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI) for information to make Application Signals work. We've also identified other considerations that you should keep in mind when enabling Python applications for Application Signals. For more information, see Python application doesn't start after Application Signals is enabled.

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

For the ESM module format, enabling Application Signals through the console or by annotating the manifest file doesn't work. Skip step 6 of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. Add the following environmental variables to the Dockerfile for your application and build the image.

```
...
ENV OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
ENV OTEL_TRACES_SAMPLER_ARG='endpoint=http://cloudwatch-agent.amazon-
cloudwatch:2000'
ENV OTEL_TRACES_SAMPLER='xray'
ENV OTEL_EXPORTER_OTLP_PROTOCOL='http/protobuf'
ENV OTEL_EXPORTER_OTLP_TRACES_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/traces'
ENV OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/metrics'
ENV OTEL_METRICS_EXPORTER='none'
ENV OTEL_LOGS_EXPORTER='none'
ENV NODE_OPTIONS='--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
ENV OTEL_SERVICE_NAME='YOUR_SERVICE_NAME' #replace with a proper service name
ENV OTEL_PROPAGATORS='tracecontext,baggage,b3,xray'
...

# command to start the application
# for example
# CMD ["node", "index.mjs"]
```

3. Add the environmental variables OTEL_RESOURCE_ATTRIBUTES_POD_NAME, OTEL_RESOURCE_ATTRIBUTES_NODE_NAME, OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME, POD_NAMESPACE and OTEL_RESOURCE_ATTRIBUTES to the deployment yaml file for the application. For example:

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: nodejs-app
  labels:
    app: nodejs-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
      # annotations:
      # make sure this annotation doesn't exist
      #   instrumentation.opentelemetry.io/inject-nodejs: 'true'
    spec:
      containers:
      - name: nodejs-app
        image: your-nodejs-application-image #replace with a proper image uri
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
          - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.labels['app'] # Assuming 'app' label is set to
 the deployment name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: OTEL_RESOURCE_ATTRIBUTES
            value: "k8s.deployment.name=
$(OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME),k8s.namespace.name=
```

```
$(POD_NAMESPACE),k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=
$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME)"
```

4. Deploy the Node.js application to the cluster.

After you have enabled your applications on the Amazon EKS Clusters, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable Application Signals on an Amazon EKS cluster using the CloudWatch Observability add-on advanced configuration

By default, OpenTelemetry (OTEL) based Application Performance Monitoring (APM) is enabled through Application Signals when installing either the CloudWatch Observability EKS add-on (V5.0.0 or greater) or the Helm chart. You can further customize specific settings using the advanced configuration for the Amazon EKS add-on or by overriding values with the Helm chart.

> **Note**
>
> If you use any OpenTelemetry (OTEL) based APM solution, enabling Application Signals affects your existing observability setup. Review your current implementation before proceeding. To maintain your existing APM setup after upgrading to V5.0.0 or later, see Opt out of Application Signals.

CloudWatch Observability Add-on also provides additional fine-grained control to include or exclude specific services as needed in the new advanced configuration. For more information, see Enabling APM through Application Signals for your Amazon EKS cluster .

Enable Application Signals on Amazon EKS using AWS CDK

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. See Enable Application Signals in your account.

1. Enable Application Signals for your applications.

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
```

```
  'ApplicationSignalsServiceRole', { }
);
```

The Discovery CloudFormation resource grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Install the `amazon-cloudwatch-observability` add-on.

   - Create an IAM role with the `CloudWatchAgentServerPolicy` and the OIDC associated with the cluster.

   ```
   const cloudwatchRole = new Role(this, 'CloudWatchAgentAddOnRole', {
       assumedBy: new OpenIdConnectPrincipal(cluster.openIdConnectProvider),
       managedPolicies:
    [ManagedPolicy.fromAwsManagedPolicyName('CloudWatchAgentServerPolicy')],
   });
   ```

3. Install the add-on with the IAM role created above.

```
new CfnAddon(this, 'CloudWatchAddon', {
    addonName: 'amazon-cloudwatch-observability',
    clusterName: cluster.clusterName,
    serviceAccountRoleArn: cloudwatchRole.roleArn
});
```

4. Add one of the following into the `PodTemplate` section of your workload manifest file.

Language	File
Java	instrumentation.opentelemetry.io/inject-java: "true"
Python	instrumentation.opentelemetry.io/inject-python: "true"
.Net	instrumentation.opentelemetry.io/inject-dotnet: "true"
Node.js	instrumentation.opentelemetry.io/inject-nodejs: "true"

```
const deployment = {
  apiVersion: "apps/v1",
  kind: "Deployment",
  metadata: { name: "sample-app" },
  spec: {
    replicas: 3,
    selector: {
      matchLabels: {
        "app": "sample-app"
      }
    },
    template: {
      metadata: {
        labels: {
          "app": "sample-app"
        },
        annotations: {
          "instrumentation.opentelemetry.io/inject-$LANG": "true"
        }
      },
      spec: {...},
    },
  },
};
```

```
cluster.addManifest('sample-app', deployment)
```

Enable Application Signals on Amazon EKS using Model Context Protocol (MCP)

You can use the CloudWatch Application Signals Model Context Protocol (MCP) server to enable Application Signals on your Amazon EKS clusters through conversational AI interactions. This provides a natural language interface for setting up Application Signals monitoring.

The MCP server automates the enablement process by understanding your requirements and generating the appropriate configuration. Instead of manually following console steps or writing CDK code, you can simply describe what you want to enable.

Prerequisites

Before using the MCP server to enable Application Signals, make sure you have:

- A Development Environment that supports MCP (such as Kiro, Claude Desktop, VSCode with MCP extensions, or other MCP-compatible tools)
- The CloudWatch Application Signals MCP server configured in your IDE. For detailed setup instructions, see CloudWatch Application Signals MCP Server documentation.

Using the MCP server

After you have configured the CloudWatch Application Signals MCP server in your IDE, you can request enablement guidance using natural language prompts. While the coding assistant can infer context from your project structure, providing specific details in your prompts helps make sure that you receive more accurate and relevant guidance. Include information such as your application language, Amazon EKS cluster name, and absolute paths to your infrastructure and application code.

Best practice prompts (specific and complete):

```
"Enable Application Signals for my Python service running on EKS.
My app code is in /home/user/flask-api and IaC is in /home/user/flask-api/terraform"

"I want to add observability to my Node.js application on EKS cluster 'production-
cluster'.
The application code is at /Users/dev/checkout-service and
the Kubernetes manifests are at /Users/dev/checkout-service/k8s"
```

```
"Help me instrument my Java Spring Boot application on EKS with Application Signals.
Application directory: /opt/apps/payment-api
CDK infrastructure: /opt/apps/payment-api/cdk"
```

Less effective prompts:

```
"Enable monitoring for my app"
# Missing: platform, language, paths

"Enable Application Signals. My code is in ./src and IaC is in ./infrastructure"
# Problem: Relative paths instead of absolute paths

"Enable Application Signals for my EKS service at /home/user/myapp"
# Missing: programming language
```

Quick template:

```
"Enable Application Signals for my [LANGUAGE] service on EKS.
App code: [ABSOLUTE_PATH_TO_APP]
IaC code: [ABSOLUTE_PATH_TO_IAC]"
```

Benefits of using the MCP server

Using the CloudWatch Application Signals MCP server offers several advantages:

- **Natural language interface:** Describe what you want to enable without memorizing commands or configuration syntax
- **Context-aware guidance:** The MCP server understands your specific environment and provides tailored recommendations
- **Reduced errors:** Automated configuration generation minimizes manual typing errors
- **Faster setup:** Get from intention to implementation more quickly
- **Learning tool:** See the generated configurations and understand how Application Signals works

Additional resources

For more information about configuring and using the CloudWatch Application Signals MCP server, see the MCP server documentation.

Enable your applications on Amazon EC2

Enable CloudWatch Application Signals on Amazon EC2 by using the custom setup steps described in this section.

For applications running on Amazon EC2, you install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

The instructions in this section are for Java, Python, and .NET applications. The steps have been tested on Amazon EC2 instances, but are also expected to work on other architectures that support AWS Distro for OpenTelemetry.

Requirements

- To get support for Application Signals, you must use the most recent version of both the CloudWatch agent and the AWS Distro for OpenTelemetry agent.
- You must have the AWS CLI installed on the instance. We recommend AWS CLI version 2, but version 1 should also work. For more information about installing the AWS CLI, see Install or update the latest version of the AWS CLI.

> **Important**
>
> If you are already using OpenTelemetry with an application that you intend to enable for Application Signals, see Supported systems before you enable Application Signals.

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Download and start the CloudWatch agent

To install the CloudWatch agent as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the CloudWatch agent to the instance. If the instance already has the CloudWatch agent installed, you might need to update it. Only versions of the agent released on November 30, 2023 or later support CloudWatch Application Signals.
2. Before you start the CloudWatch agent, configure it to enable Application Signals. The following example is a CloudWatch agent configuration that enables Application Signals for both metrics and traces on an EC2 host.

   We recommend that you place this file at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json` on Linux systems.

   ```
   {
     "traces": {
       "traces_collected": {
         "application_signals": {}
       }
     },
     "logs": {
       "metrics_collected": {
         "application_signals": {}
       }
     }
   }
   ```

3. Attach the **CloudWatchAgentServerPolicy** IAM policy to the IAM role of your Amazon EC2 instance. For permissions for on-premises hosts, see Permissions for on-premises servers.

   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

   b. Choose **Roles** and find the role used by your Amazon EC2 instance. Then choose the name of that role.

   c. In the **Permissions** tab, choose **Add permissions**, **Attach policies**.

   d. Find **CloudWatchAgentServerPolicy**. Use the search box if needed. Then select the check box for that policy and choose **Add permissions**.

4. Start the CloudWatch agent by entering the following commands. Replace *agent-config-file-path* with the path to the CloudWatch agent configuration file, such as `./amazon-cloudwatch-agent.json`. You must include the `file:` prefix as shown.

```
export CONFIG_FILE_PATH=./amazon-cloudwatch-agent.json
```

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl \
-a fetch-config \
-m ec2 -s -c file:agent-config-file-path
```

Permissions for on-premises servers

For an on-premises host, you will need to provide AWS authorization to your device.

To set up permissions for an on-premises host

1. Create the IAM user to be used to provide permissions to your on-premises host:
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. Choose **Users**, **Create User**.
   c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for AWS that will be used to authenticate your host. Then choose **Next**
   d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.
   e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.
   f. On the **Review and create** page, make sure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.
   g. Choose **Create user**
2. Create and retrieve your AWS access key and secret key:
   a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.
   b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.
   c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.
   d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.

e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.

f. Choose **Done**.

3. Configure your AWS credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step.

```
$ aws configure
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

Step 3: Instrument your application and start it

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your Java applications as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the AWS Distro for OpenTelemetry Java auto-instrumentation agent. You can download the latest version by using this link. You can view information about all released versions at aws-otel-java-instrumentation Releases.
2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   - For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

     - (Optional) `service.name` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.
     - (Optional) `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application

in Application Signals dashboards. If you don't specify this, one of the following defaults is used:

- If this is an instance that is part of an Auto Scaling group, it is set to `ec2:`*`name-of-Auto-Scaling-group`*
- If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
- If this is an on-premises host, it is set to `generic:default`

This environment variable is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

- For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`
- For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`
- For the `JAVA_TOOL_OPTIONS` variable, specify the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored.

  ```
  export JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH"
  ```

  For example:

  ```
  export AWS_ADOT_JAVA_INSTRUMENTATION_PATH=./aws-opentelemetry-agent.jar
  ```

- For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to none. This disables other metrics exporters so that only the Application Signals exporter is used.
- Set `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` to `true`. This generates Application Signals metrics from traces.

3. Start your application with the environment variables listed in the previous step. The following is an example of a starting script.

> **Note**
>
> The following configuration supports only versions 1.32.2 and later of the AWS Distro for OpenTelemetry auto-instrumentation agent for Java.

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME" \
java -jar $MY_JAVA_APP.jar
```

4. (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example of a starting script that helps enable log correlation.

```
JAVA_TOOL_OPTIONS=" -javaagent:$AWS_ADOT_JAVA_INSTRUMENTATION_PATH" \
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORT=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
```

```
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_JAVA_APP.jar
```

Python

> **Note**
>
> If you're using a pre-fork server (WSGI or ASGI) such as Gunicorn, uWSGI, or Uvicorn for your Python application, in addition to the following steps in this section, see No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI) for information to make Application Signals work.

To instrument your Python applications as part of enabling Application Signals on an Amazon EC2 instance

1. Download the latest version of the AWS Distro for OpenTelemetry Python auto-instrumentation agent. Install it by running the following command.

   ```
   pip install aws-opentelemetry-distro
   ```

   You can view information about all released versions at AWS Distro for OpenTelemetry Python instrumentation.

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   a. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

      - `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
  - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:`*`name-of-Auto-Scaling-group`*.
  - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
  - If this is an on-premises host, it is set to `generic:default`

  This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

b. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, specify `http/protobuf` to export telemetry data over HTTP to the CloudWatch agent endpoints listed in the following steps.

c. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`

d. For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`

e. For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to none. This disables other metrics exporters so that only the Application Signals exporter is used.

f. Set the `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` variable to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.

3. Start your application with the environment variables discussed in the previous step. The following is an example of a starting script.

- Replace $SVC_NAME with your application name. This will be displayed as the name of the application, in Application Signals dashboards.
- Replace $PYTHON_APP with the location and name of your application.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
opentelemetry-instrument python $MY_PYTHON_APP.py
```

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, make sure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

4. (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be

correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

The following is an example of a starting script that helps enable log correlation.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_PYTHON_DISTRO=aws_distro \
OTEL_PYTHON_CONFIGURATOR=aws_configurator \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$
 \
java -jar $MY_PYTHON_APP.jar
```

.NET

To instrument your .NET applications as part of enabling Application Signals on an Amazon EC2 instance or on-premises host

1. Download the latest version of the AWS Distro for OpenTelemetry .NET auto-instrumentation package. You can download the latest version at aws-otel-dotnet-instrumentation Releases.
2. To enable Application Signals, set the following environment variables to provide additional information before you start your application. These variables are necessary to set up the startup hook for .NET instrumentation, before you start your .NET application. Replace `dotnet-service-name` in the `OTEL_RESOURCE_ATTRIBUTES` environment variable with the service name of your choice.

- The following is an example for Linux.

```
export INSTALL_DIR=OpenTelemetryDistribution
export CORECLR_ENABLE_PROFILING=1
export CORECLR_PROFILER={918728DD-259F-4A6A-AC2B-B85E1B658318}
export CORECLR_PROFILER_PATH=${INSTALL_DIR}/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so
export DOTNET_ADDITIONAL_DEPS=${INSTALL_DIR}/AdditionalDeps
export DOTNET_SHARED_STORE=${INSTALL_DIR}/store
export DOTNET_STARTUP_HOOKS=${INSTALL_DIR}/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll
export OTEL_DOTNET_AUTO_HOME=${INSTALL_DIR}

export
 OTEL_DOTNET_AUTO_PLUGINS="AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_ENDPOINT=http://127.0.0.1:4316
export OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://127.0.0.1:4316/
v1/metrics
export OTEL_METRICS_EXPORTER=none
export OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
export OTEL_TRACES_SAMPLER=xray
export OTEL_TRACES_SAMPLER_ARG=http://127.0.0.1:2000
```

- The following is an example for Windows Server.

```
$env:INSTALL_DIR = "OpenTelemetryDistribution"
$env:CORECLR_ENABLE_PROFILING = 1
$env:CORECLR_PROFILER = "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
$env:CORECLR_PROFILER_PATH = Join-Path $env:INSTALL_DIR "win-x64/
OpenTelemetry.AutoInstrumentation.Native.dll"
$env:DOTNET_ADDITIONAL_DEPS = Join-Path $env:INSTALL_DIR "AdditionalDeps"
$env:DOTNET_SHARED_STORE = Join-Path $env:INSTALL_DIR "store"
$env:DOTNET_STARTUP_HOOKS = Join-Path $env:INSTALL_DIR "net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
$env:OTEL_DOTNET_AUTO_HOME = $env:INSTALL_DIR
```

```
$env:OTEL_DOTNET_AUTO_PLUGINS =
 "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"

$env:OTEL_RESOURCE_ATTRIBUTES = "service.name=dotnet-service-name"
$env:OTEL_EXPORTER_OTLP_PROTOCOL = "http/protobuf"
$env:OTEL_EXPORTER_OTLP_ENDPOINT = "http://127.0.0.1:4316"
$env:OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT = "http://127.0.0.1:4316/
v1/metrics"
$env:OTEL_METRICS_EXPORTER = "none"
$env:OTEL_AWS_APPLICATION_SIGNALS_ENABLED = "true"
$env:OTEL_TRACES_SAMPLER = "xray"
$env:OTEL_TRACES_SAMPLER_ARG = "http://127.0.0.1:2000"
```

3. Start your application with the environment variables listed in the previous step.

   (Optional) Alternatively, you can use the installation scripts provided to help installation and setup of AWS Distro for OpenTelemetry .NET auto-instrumentation package.

   For Linux, download and install the Bash installation script from the GitHub releases page:

   ```
   # Download and Install
   curl -L -O https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
   releases/latest/download/aws-otel-dotnet-install.sh
   chmod +x ./aws-otel-dotnet-install.sh
   ./aws-otel-dotnet-install.sh

   # Instrument
   . $HOME/.otel-dotnet-auto/instrument.sh
   export OTEL_RESOURCE_ATTRIBUTES=service.name=dotnet-service-name
   ```

   For Windows Server, download and install the PowerShell installation script from the GitHub releases page:

   ```
   # Download and Install
   $module_url = "https://github.com/aws-observability/aws-otel-dotnet-
   instrumentation/releases/latest/download/AWS.Otel.DotNet.Auto.psm1"
   $download_path = Join-Path $env:temp "AWS.Otel.DotNet.Auto.psm1"
   Invoke-WebRequest -Uri $module_url -OutFile $download_path
   Import-Module $download_path
   Install-OpenTelemetryCore
   ```

```
# Instrument
Import-Module $download_path
Register-OpenTelemetryForCurrentSession -OTelServiceName "dotnet-service-name"
Register-OpenTelemetryForIIS
```

You can find the NuGet package of the AWS Distro for OpenTelemetry .NET auto-instrumentation package in the official NuGet repository. Be sure to check the README file for instructions.

Node.js

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your Node.js applications as part of enabling Application Signals on an Amazon EC2 instance

1. Download the latest version of the AWS Distro for OpenTelemetry JavaScript auto-instrumentation agent for Node.js. Install it by running the following command.

   ```
   npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
   ```

   You can view information about all released versions at AWS Distro for OpenTelemetry JavaScript instrumentation.

2. To optimize your Application Signals benefits, use environment variables to provide additional information before you start your application. This information will be displayed in Application Signals dashboards.

   a. For the `OTEL_RESOURCE_ATTRIBUTES` variable, specify the following information as key-value pairs:

      - `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used.

- `deployment.environment` sets the environment that the application runs in. This will be diplayed as the **Hosted In** environment of your application in Application Signals dashboards. If you don't specify this, one of the following defaults is used:
  - If this is an instance that is part of an Auto Scaling group, it is set to `ec2:`*`name-of-Auto-Scaling-group`*.
  - If this is an Amazon EC2 instance that is not part of an Auto Scaling group, it is set to `ec2:default`
  - If this is an on-premises host, it is set to `generic:default`

  This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.

b. For the `OTEL_EXPORTER_OTLP_PROTOCOL` variable, specify `http/protobuf` to export telemetry data over HTTP to the CloudWatch agent endpoints listed in the following steps.

c. For the `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` variable, specify the base endpoint URL where traces are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces`

d. For the `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT` variable, specify the base endpoint URL where metrics are to be exported to. The CloudWatch agent exposes 4316 as its OTLP port over HTTP. On Amazon EC2, because applications communicate with the local CloudWatch agent, you should set this value to `OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics`

e. For the `OTEL_METRICS_EXPORTER` variable, we recommend that you set the value to none. This disables other metrics exporters so that only the Application Signals exporter is used.

f. Set the `OTEL_AWS_APPLICATION_SIGNALS_ENABLED` variable to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.

3. Start your application with the environment variables discussed in the previous step. The following is an example of a starting script.

- Replace $SVC_NAME with your application name. This will be displayed as the name of the application, in Application Signals dashboards.

```
OTEL_METRICS_EXPORTER=none \
OTEL_LOGS_EXPORTER=none \
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
OTEL_TRACES_SAMPLER=xray \
OTEL_TRACES_SAMPLER_ARG="endpoint=http://localhost:2000" \
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/metrics
 \
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
OTEL_RESOURCE_ATTRIBUTES="service.name=$SVC_NAME" \
node --require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/
register' your-application.js
```

4. (Optional) To enable log correlation, in `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example of a starting script that helps enable log correlation.

```
export OTEL_METRICS_EXPORTER=none \
export OTEL_LOGS_EXPORTER=none \
export OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true \
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf \
export OTEL_TRACES_SAMPLER=xray \
export OTEL_TRACES_SAMPLER_ARG=endpoint=http://localhost:2000 \
export OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT=http://localhost:4316/v1/
metrics \
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4316/v1/traces \
```

```
export
 OTEL_RESOURCE_ATTRIBUTES="aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=
 \
node --require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/
register' your-application.js
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

To enable Application Signals for a Node.js application with ESM, you need to modify the steps in the previous procedure.

First, install `@opentelemetry/instrumentation` for your Node.js application:

```
npm install @opentelemetry/instrumentation@0.54.0
```

Then, in steps 3 and 4 in the previous procedure, change the node options from:

```
--require '@aws/aws-distro-opentelemetry-node-autoinstrumentation/register'
```

to the following:

```
--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/register --
experimental-loader=@opentelemetry/instrumentation/hook.mjs
```

Enable Application Signals on Amazon EC2 using Model Context Protocol (MCP)

You can use the CloudWatch Application Signals Model Context Protocol (MCP) server to enable Application Signals on your Amazon EC2 instances through conversational AI interactions. This provides a natural language interface for setting up Application Signals monitoring.

The MCP server automates the enablement process by understanding your requirements and generating the appropriate configuration. Instead of manually following setup steps, you can simply describe what you want to enable.

Prerequisites

Before using the MCP server to enable Application Signals, ensure you have:

- A Development Environment that supports MCP (such as Kiro, Claude Desktop, VSCode with MCP extensions, or other MCP-compatible tools)
- The CloudWatch Application Signals MCP server configured in your IDE. For detailed setup instructions, see CloudWatch Application Signals MCP Server documentation.

Using the MCP server

Once you have configured the CloudWatch Application Signals MCP server in your IDE, you can request enablement guidance using natural language prompts. While the coding assistant can infer context from your project structure, providing specific details in your prompts helps ensure more accurate and relevant guidance. Include information such as your application language, instance details, and absolute paths to your infrastructure and application code.

Best practice prompts (specific and complete):

```
"Enable Application Signals for my Python service running on EC2.
My app code is in /home/ec2-user/flask-api and IaC is in /home/ec2-user/flask-api/
terraform"

"I want to add observability to my Java application on EC2.
The application code is at /opt/apps/checkout-service and
the infrastructure code is at /opt/apps/checkout-service/cloudformation"

"Help me instrument my Node.js application on EC2 with Application Signals.
Application directory: /home/ubuntu/payment-api
Terraform code: /home/ubuntu/payment-api/terraform"
```

Less effective prompts:

```
"Enable monitoring for my app"
# Missing: platform, language, paths

"Enable Application Signals. My code is in ./src and IaC is in ./infrastructure"
# Problem: Relative paths instead of absolute paths

"Enable Application Signals for my EC2 service at /home/user/myapp"
# Missing: programming language
```

Quick template:

```
"Enable Application Signals for my [LANGUAGE] service on EC2.
App code: [ABSOLUTE_PATH_TO_APP]
IaC code: [ABSOLUTE_PATH_TO_IAC]"
```

Benefits of using the MCP server

Using the CloudWatch Application Signals MCP server offers several advantages:

- **Natural language interface:** Describe what you want to enable without memorizing commands or configuration syntax
- **Context-aware guidance:** The MCP server understands your specific environment and provides tailored recommendations
- **Reduced errors:** Automated configuration generation minimizes manual typing errors
- **Faster setup:** Get from intention to implementation more quickly
- **Learning tool:** See the generated configurations and understand how Application Signals works

Additional resources

For more information about configuring and using the CloudWatch Application Signals MCP server, see the MCP server documentation.

(Optional) Monitor your application health

After you have enabled your applications on Amazon EC2, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable your applications on Amazon ECS

Enable CloudWatch Application Signals on Amazon ECS by using the custom setup steps described in this section.

For applications running on Amazon ECS, you install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Use a custom setup to enable Application Signals on Amazon ECS

Use these custom setup instructions to onboard your applications on Amazon ECS to CloudWatch Application Signals. You install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself.

There are two methods for deploying Application Signals on Amazon ECS. Choose the one that is best for your environment.

- Deploy using the sidecar strategy – You add a CloudWatch agent sidecar container to each task definition in the cluster.

  Advantages:

  - Supports both the `ec2` and `Fargate` launch types.
  - You can always use `localhost` as the IP address when you set up environment variables.

  Disadvantages:

  - You must set up the CloudWatch agent sidecar container for each service task that runs in the cluster.
  - Only the `awsvpc` network mode is supported.
- Deploy using the daemon strategy – You add a CloudWatch agent task only once in the cluster, and the Amazon ECS daemon scheduling strategy deploys it as needed. The ensures that each instance continuously receives traces and metrics, providing centralized visibility without the need for the agent to run as a sidecar with each application task definition.

  Advantages:

  - You need to set up the daemon service for the CloudWatch agent only once in the cluster.

  Disadvantages:

  - Doesn't support the Fargate launch type.
  - If you use the `awsvpc` or `bridge` network mode, you have to manually specify each container instance's private IP address in the environment variables.

With either method, on Amazon ECS clusters Application Signals doesn't autodiscover the names of your services. You must specify your service names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Deploy using the sidecar strategy

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Create IAM roles

You must create an IAM role. If you already have created this role, you might need to add permissions to it.

- **ECS task role—** Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy**.

For more information about creating IAM roles, see Creating IAM Roles.

Step 3: Prepare CloudWatch agent configuration

First, prepare the agent configuration with Application Signals enabled. To do this, create a local file named `/tmp/ecs-cwagent.json`.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

Then upload this configuration to the SSM Parameter Store. To do this, enter the following command. In the file, replace *$REGION* with your actual Region name.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
```

```
--region "$REGION"
```

Step 4: Instrument your application with the CloudWatch agent

The next step is to instrument your application for CloudWatch Application Signals.

Java

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

   ```
   "volumes": [
     {
       "name": "opentelemetry-auto-instrumentation"
     }
   ]
   ```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

   ```
   {
     "name": "ecs-cwagent",
     "image": "$IMAGE",
     "essential": false,
     "secrets": [
       {
         "name": "CW_CONFIG_CONTENT",
         "valueFrom": "ecs-cwagent"
       }
     ],
     "logConfiguration": {
       "logDriver": "awslogs",
       "options": {
         "awslogs-create-group": "true",
         "awslogs-group": "/ecs/ecs-cwagent",
   ```

```
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "/javaagent.jar",
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container. You must be using version 1.32.2 or later of the AWS Distro for OpenTelemetry auto-instrumentation agent for Java.

Environment variable	Setting to enable Application Signals
OTEL_RESOURCE_ATTRIBUTES	Specify the following information as key-value pairs: • service.name sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of UnknownService is used • deployment.environment parameter defines your application's runtime environment and controls the Environment dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Applicati on Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additiona l environment variable aws.log.g roup.names to be the log group name for your application log. By doing so, the traces and metrics from your applicati on can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names

Environment variable	Setting to enable Application Signals
	for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Set to `http://localhost:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Set to `http://localhost:4316/v1/traces` to send traces to the CloudWatch sidecar.

Environment variable	Setting to enable Application Signals
OTEL_TRACES_SAMPLER	Set this to xray to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Set xray as one of the propagators.
JAVA_TOOL_OPTIONS	Set to " -javaagent:$ *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* " Replace *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* with the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example, /otel-auto-instrumentation/javaagent.jar

6. Mount the volume opentelemetry-auto-instrumentation that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Java application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
```

```
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, make sure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.
- If you're using a pre-fork server (WSGI or ASGI) such as Gunicorn, uWSGI, or Uvicorn for your Python application, in addition to the following steps in this section, see No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI) for information to make Application Signals work.

To instrument your Python application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": false,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
```

```
        }
    ]
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Applicati on Signals, and is converted into X-Ray

Environment variable	Setting to enable Application Signals
	trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additiona l environment variable `aws.log.g roup.names` to be the log group name for your application log. By doing so, the traces and metrics from your applicati on can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_ LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-gro up-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNAL S_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to `none` to disable other metrics exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to CloudWatch using HTTP.

Environment variable	Setting to enable Application Signals
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	Set to http://127.0.0.1:4316/v1/metrics to send metrics to the CloudWatch sidecar.
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	Set to http://127.0.0.1:4316/v1/traces to send traces to the CloudWatch sidecar.
OTEL_TRACES_SAMPLER	Set this to xray to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Add xray as one of the propagators.
OTEL_PYTHON_DISTRO	Set to aws_distro to use the ADOT Python instrumentation.
OTEL_PYTHON_CONFIGURATOR	Set to aws_configurator to use the ADOT Python configuration.
PYTHONPATH	Replace $APP_PATH with the location of the application's working directory within the container. This is required for the Python interpreter to find your application modules.
DJANGO_SETTINGS_MODULE	Required only for Django applications. Set it to the location of your Django application's settings.py file. Replace $PATH_TO_SETTINGS .

6. Mount the volume opentelemetry-auto-instrumentation-python that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Python application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
```

```
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
```

```
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

7. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example. To enable log correlation, use this example when you mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
  ...
  "environment": [
```

```
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
```

```
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

.NET

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": false,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

   For a Linux container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "cp",
      "-a",
      "autoinstrumentation/.",
      "/otel-auto-instrumentation"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "/otel-auto-instrumentation",
          "readOnly": false
      }
```

```
  ]
}
```

For a Windows Server container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "CMD",
      "/c",
      "xcopy",
      "/e",
      "C:\\autoinstrumentation\\*",
      "C:\\otel-auto-instrumentation",
      "&&",
      "icacls",
      "C:\\otel-auto-instrumentation",
      "/grant",
      "*S-1-1-0:R",
      "/T"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "C:\\otel-auto-instrumentation",
          "readOnly": false
      }
  ]
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
    {
        "containerName": "init",
        "condition": "SUCCESS"
    }
]
```

5. Add the following environment variables to your application container. You must be using version 1.1.0 or later of the AWS Distro for OpenTelemetry auto-instrumentation agent for .NET.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.

Environment variable	Setting to enable Application Signals
OTEL_LOGS_EXPORTER	Set to none to disable other logs exporters.
OTEL_EXPORTER_OTLP_PROTOCOL	Set to http/protobuf to send metrics and traces to Application Signals using HTTP.
OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT	Set to http://localhost:4316/v1/metrics to send metrics to the CloudWatch sidecar.
OTEL_EXPORTER_OTLP_ENDPOINT	Set to http://localhost:4316/ to send traces to the CloudWatch sidecar.
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	Set to http://localhost:4316/v1/traces to send traces to the CloudWatch sidecar.
OTEL_DOTNET_AUTO_HOME	Set to the installation location of ADOT .NET automatic instrumentation.
OTEL_DOTNET_AUTO_PLUGINS	Set to AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin, AWS.Distro.OpenTelemetry.AutoInstrumentation to enable the Application Signals plugin.
CORECLR_ENABLE_PROFILING	Set to 1 to enable the profiler.
CORECLR_PROFILER	Set to {918728DD-259F-4A6A-AC2B-B85E1B658318} as the CLSID of the profiler.

Environment variable	Setting to enable Application Signals
CORECLR_PROFILER_PATH	Set this to the path of the profiler. On Linux, set it to ${OTEL_DOTNET_AUTO_HOME}/linux-x64/OpenTelemetry.AutoInstrumentation.Native.so On Windows Server, set it to ${OTEL_DOTNET_AUTO_HOME}/win-x64/OpenTelemetry.AutoInstrumentation.Native.dll
DOTNET_ADDITIONAL_DEPS	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/AdditionalDeps .
DOTNET_SHARED_STORE	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/store .
DOTNET_STARTUP_HOOKS	Set this to path of the managed assembly ${OTEL_DOTNET_AUTO_HOME}/net/OpenTelemetry.AutoInstrumentation.StartupHook.dll to run before the main application's entry point.

6. Mount the volume opentelemetry-auto-instrumentation that you defined in step 1 of this procedure. For Linux, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
        {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
```

```
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "/otel-auto-instrumentation/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "/otel-auto-instrumentation/AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "/otel-auto-instrumentation/store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "/otel-auto-instrumentation/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "/otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=aws-dotnet-service-name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
```

```
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
            "name": "OTEL_TRACES_SAMPLER_ARG",
            "value": "endpoint=http://localhost:2000"
       },
        {
            "name": "OTEL_PROPAGATORS",
            "value": "tracecontext,baggage,b3,xray"
        }
    ],
    "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
```

```
            "containerPath": "/otel-auto-instrumentation",
            "readOnly": false
        }
    ]
}
```

For Windows Server, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
       {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "C:\\otel-auto-instrumentation\\win-x64\
\OpenTelemetry.AutoInstrumentation.Native.dll"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "C:\\otel-auto-instrumentation\\AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "C:\\otel-auto-instrumentation\\store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "C:\\otel-auto-instrumentation\\net\
\OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
```

```
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "C:\\otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value": "aws.log.group.names=
$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://localhost:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://localhost:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
```

```
        },
        {
            "name": "OTEL_TRACES_SAMPLER_ARG",
            "value": "endpoint=http://localhost:2000"
        },
         {
             "name": "OTEL_PROPAGATORS",
             "value": "tracecontext,baggage,b3,xray"
         }
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

Node.js

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-node"
```

```
  }
]
```

2. Add a CloudWatch agent sidecar definition. To do this, append a new container called `ecs-cwagent` to your application's task definition. Replace *$REGION* with your actual Region name. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

   If you want to enable the CloudWatch agent with a daemon strategy instead, see the instructions at Deploy using the daemon strategy.

```
{
  "name": "ecs-cwagent",
  "image": "$IMAGE",
  "essential": false,
  "secrets": [
    {
      "name": "CW_CONFIG_CONTENT",
      "valueFrom": "ecs-cwagent"
    }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-create-group": "true",
      "awslogs-group": "/ecs/ecs-cwagent",
      "awslogs-region": "$REGION",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
```

3. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "-a",
```

```
    "/autoinstrumentation/.",
    "/otel-auto-instrumentation-node"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
}
```

4. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

5. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon ECS Application metrics in CloudWatch.

Environment variable	Setting to enable Application Signals
	When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using OTLP/HTTP and protobuf.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Set to `http://localhost:4316/v1/metrics` to send metrics to the CloudWatch sidecar.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Set to `http://localhost:4316/v1/traces` to send traces to the CloudWatch sidecar.
`OTEL_TRACES_SAMPLER`	Set this to `xray` to set X-Ray as the traces sampler.
`OTEL_PROPAGATORS`	Set `xray` as one of the propagators.
`NODE_OPTIONS`	Set to `--require` *`AWS_ADOT_NODE_INSTRUMENTATION_PATH`*. Replace *`AWS_ADOT_NODE_INSTRUMENTATION_PATH`* with the path where the AWS Distro for OpenTelemetry Node.js auto-instrumentation is stored. For example, `/otel-auto-instrumentation-node/autoinstrumentation.js`

6. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Node.js application. If you want to enable log correlation, see the next step instead.

   For your Application Container, add a dependency on the `init` container to make sure that container finishes before your application container starts.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
```

```
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "NODE_OPTIONS",
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

7. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example. Use this example to enable log correlation when you mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://localhost:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://localhost:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://localhost:2000"
    },
    {
      "name": "NODE_OPTIONS",
```

```
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

For the ESM module format, using the `init` container to inject the Node.js instrumentation SDK doesn't apply. To enable Application Signals for Node.js with ESM, skip steps 1 and 3 of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. In steps 5 and 6 in the previous procedure, remove the mounting of the volume `opentelemetry-auto-instrumentation-node`:

```
 "mountPoints": [
    {
```

```
        "sourceVolume": "opentelemetry-auto-instrumentation-node",
        "containerPath": "/otel-auto-instrumentation-node",
        "readOnly": false
    }
 ]
```

Replace the node options with the following.

```
{
    "name": "NODE_OPTIONS",
    "value": "--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs"
}
```

Step 5: Deploy your application

Create a new revision of your task definition and deploy it to your application cluster. You should see three containers in the newly created task:

- `init`– A required container for initializing Application Signals.
- `ecs-cwagent`– A container running the CloudWatch agent
- *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

(Optional) Step 6: Monitor your application health

After you have enabled your applications on Amazon ECS, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Deploy using the daemon strategy

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Create IAM roles

You must create an IAM role. If you already have created this role, you might need to add permissions to it.

- **ECS task role—** Containers use this role to run. The permissions should be whatever your applications need, plus **CloudWatchAgentServerPolicy**.

For more information about creating IAM roles, see Creating IAM Roles.

Step 3: Prepare CloudWatch agent configuration

First, prepare the agent configuration with Application Signals enabled. To do this, create a local file named `/tmp/ecs-cwagent.json`.

```
{
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {}
    }
  }
}
```

Then upload this configuration to the SSM Parameter Store. To do this, enter the following command. In the file, replace *$REGION* with your actual Region name.

```
aws ssm put-parameter \
--name "ecs-cwagent" \
--type "String" \
--value "`cat /tmp/ecs-cwagent.json`" \
--region "$REGION"
```

Step 4: Deploy the CloudWatch agent daemon service

Create the following task definition and deploy it to your application cluster. Replace *$REGION* with your actual Region name. Replace *$TASK_ROLE_ARN* and *$EXECUTION_ROLE_ARN* with the

IAM roles you prepared in Step 2: Create IAM roles. Replace *$IMAGE* with the path to the latest CloudWatch container image on Amazon Elastic Container Registry. For more information, see cloudwatch-agent on Amazon ECR.

> **Note**
>
> The daemon service exposes two ports on the host, with 4316 used as endpoint for receiving metrics and traces and 2000 as the CloudWatch trace sampler endpoint. This setup allows the agent to collect and transmit telemetry data from all application tasks running on the host. Make sure that these ports are not used by other services on the host to avoid conflicts.

```
{
  "family": "ecs-cwagent-daemon",
  "taskRoleArn": "$TASK_ROLE_ARN",
  "executionRoleArn": "$EXECUTION_ROLE_ARN",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "ecs-cwagent",
      "image": "$IMAGE",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 4316,
          "hostPort": 4316
        },
        {
          "containerPort": 2000,
          "hostPort": 2000
        }
      ],
      "secrets": [
        {
          "name": "CW_CONFIG_CONTENT",
          "valueFrom": "ecs-cwagent"
        }
      ],
      "logConfiguration": {
        "logDriver": "awslogs",
```

```
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "/ecs/ecs-cwagent",
          "awslogs-region": "$REGION",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "requiresCompatibilities": [
    "EC2"
  ],
  "cpu": "128",
  "memory": "64"
}
```

Step 5: Instrument your application

The next step is to instrument your application for Application Signals.

Java

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

   ```
   "volumes": [
     {
       "name": "opentelemetry-auto-instrumentation"
     }
   ]
   ```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

   ```
   {
     "name": "init",
     "image": "$IMAGE",
     "essential": false,
     "command": [
       "cp",
       "/javaagent.jar",
   ```

```
    "/otel-auto-instrumentation/javaagent.jar"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container. You must be using version 1.32.2 or later of the AWS Distro for OpenTelemetry auto-instrumentation agent for Java.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon

Environment variable	Setting to enable Application Signals
	ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable `aws.log.group.names` to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to none to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/metrics, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT	Sends traces to the CloudWatch daemon container. • For applications running in host mode, set this to http://localhost:4316/v1/traces . • For applications running in bridge mode or awsvpc mode, set this to http://*CW_CONTAINER_IP* :4316/v1/traces, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
OTEL_TRACES_SAMPLER	Set this to xray to set X-Ray as the traces sampler.
OTEL_PROPAGATORS	Set xray as one of the propagators.
JAVA_TOOL_OPTIONS	Set to " -javaagent:$ *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* " Replace *AWS_ADOT_JAVA_INSTRUMENTATION_PATH* with the path where the AWS Distro for OpenTelemetry Java auto-instrumentation agent is stored. For example, /otel-auto-instrumentation/javaagent.jar

5. Mount the volume opentelemetry-auto-instrumentation that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Java application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
```

```
    ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "JAVA_TOOL_OPTIONS",
      "value": " -javaagent:/otel-auto-instrumentation/javaagent.jar"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_PROPAGATORS",
      "value": "tracecontext,baggage,b3,xray"
    }
  ],
    "dependsOn": [
```

```
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation",
      "containerPath": "/otel-auto-instrumentation",
      "readOnly": false
    }
  ]
}
```

Python

Before you enable Application Signals for your Python applications, be aware of the following considerations.

- In some containerized applications, a missing PYTHONPATH environment variable can sometimes cause the application to fail to start. To resolve this, make sure that you set the PYTHONPATH environment variable to the location of your application's working directory. This is due to a known issue with OpenTelemetry auto-instrumentation. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant.
- For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.
  - Use the `--noreload` flag to prevent automatic reloading.
  - Set the DJANGO_SETTINGS_MODULE environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.
- If you're using a pre-fork server (WSGI or ASGI) such as Gunicorn, uWSGI, or Uvicorn for your Python application, in addition to the following steps in this section, see No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI) for information to make Application Signals work.

To instrument your Python application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-python"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

```
{
    "name": "init",
    "image": "$IMAGE",
    "essential": false,
    "command": [
        "cp",
        "-a",
        "/autoinstrumentation/.",
        "/otel-auto-instrumentation-python"
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation-python",
            "containerPath": "/otel-auto-instrumentation-python",
            "readOnly": false
        }
    ]
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
OTEL_RESOURCE_ATTRIBUTES	Specify the following information as key-value pairs: • service.name sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of UnknownService is used. • deployment.environment parameter defines your application's runtime environment and controls the Environment dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Application Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additional environment variable aws.log.group.names to be the log group name for your application log. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from the log group. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names

Environment variable	Setting to enable Application Signals
	for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNALS_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to none to disable other metrics exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to CloudWatch using HTTP.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/metrics, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/traces, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_TRACES_SAMPLER`	Set this to `xray` to set X-Ray as the traces sampler.

Environment variable	Setting to enable Application Signals
OTEL_TRACES_SAMPLER_ARG	Sets the traces sampler endpoint. • For applications running in host mode, set this to http://localhost:2000 . • For applications running in bridge mode or awsvpc mode, set this to http://*CW_CONTAINER_IP* :2000, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
OTEL_PROPAGATORS	Add xray as one of the propagators.
OTEL_PYTHON_DISTRO	Set to aws_distro to use the ADOT Python instrumentation.
OTEL_PYTHON_CONFIGURATOR	Set to aws_configuration to use the ADOT Python configuration.
PYTHONPATH	Replace $APP_PATH with the location of the application's working directory within the container. This is required for the Python interpreter to find your application modules.
DJANGO_SETTINGS_MODULE	Required only for Django applications. Set it to the location of your Django application's settings.py file. Replace $PATH_TO_SETTINGS .

5. Mount the volume opentelemetry-auto-instrumentation-python that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and

traces, use the following example for a Python application. If you want to enable log correlation, see the next step instead.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
```

```
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

6. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *$YOUR_APPLICATION_LOG_GROUP* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to

log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

The following is an example. To enable log correlation, use this example when you mount the volume `opentelemetry-auto-instrumentation-python` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
  ...
  "environment": [
    {
      "name": "PYTHONPATH",
      "value": "/otel-auto-instrumentation-python/opentelemetry/instrumentation/
auto_instrumentation:$APP_PATH:/otel-auto-instrumentation-python"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_PYTHON_DISTRO",
      "value": "aws_distro"
    },
    {
      "name": "OTEL_PYTHON_CONFIGURATOR",
      "value": "aws_configurator"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
```

```
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "DJANGO_SETTINGS_MODULE",
      "value": "$PATH_TO_SETTINGS.settings"
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-python",
      "containerPath": "/otel-auto-instrumentation-python",
      "readOnly": false
    }
  ]
}
```

.NET

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

   For a Linux container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
      "cp",
      "-a",
      "autoinstrumentation/.",
      "/otel-auto-instrumentation"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "/otel-auto-instrumentation",
          "readOnly": false
      }
  ]
}
```

   For a Windows Server container instance, use the following.

```
{
  "name": "init",
  "image": "$IMAGE",
```

```
  "essential": false,
  "command": [
      "CMD",
      "/c",
      "xcopy",
      "/e",
      "C:\\autoinstrumentation\\*",
      "C:\\otel-auto-instrumentation",
      "&&",
      "icacls",
      "C:\\otel-auto-instrumentation",
      "/grant",
      "*S-1-1-0:R",
      "/T"
  ],
  "mountPoints": [
      {
          "sourceVolume": "opentelemetry-auto-instrumentation",
          "containerPath": "C:\\otel-auto-instrumentation",
          "readOnly": false
      }
  ]
}
```

3. Add a dependency on the `init` container to make sure that container finishes before your application container starts.

```
"dependsOn": [
    {
        "containerName": "init",
        "condition": "SUCCESS"
    }
]
```

4. Add the following environment variables to your application container. You must be using version 1.1.0 or later of the AWS Distro for OpenTelemetry auto-instrumentation agent for .NET.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs:

Environment variable	Setting to enable Application Signals
	• `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Applicati on Signals, and is converted into X-Ray trace annotations and CloudWatch metric dimensions.
`OTEL_AWS_APPLICATION_SIGNAL S_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to `none` to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to `none` to disable other logs exporters.
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using HTTP.

Environment variable	Setting to enable Application Signals
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/metrics, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*`CW_CONTAINER_IP`* :4316/v1/traces, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_TRACES_SAMPLER_ARG`	Sets the traces sampler endpoint. • For applications running in `host` mode, set this to `http://localhost:2000` . • For applications running in `bridge` mode or `awsvpc` mode, set this to `http://`*`CW_CONTAINER_IP`*`:2000`, where *`CW_CONTAINER_IP`* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_DOTNET_AUTO_HOME`	Set to the installation location of ADOT .NET automatic instrumentation.

Environment variable	Setting to enable Application Signals
OTEL_DOTNET_AUTO_PLUGINS	Set to AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin, AWS.Distro.OpenTelemetry.AutoInstrumentation to enable the Application Signals plugin.
CORECLR_ENABLE_PROFILING	Set to 1 to enable the profiler.
CORECLR_PROFILER	Set to {918728DD-259F-4A6A-AC2B-B85E1B658318} as the CLSID of the profiler.
CORECLR_PROFILER_PATH	Set this to the path of the profiler. On Linux, set it to ${OTEL_DOTNET_AUTO_HOME}/linux-x64/OpenTelemetry.AutoInstrumentation.Native.so On Windows Server, set it to ${OTEL_DOTNET_AUTO_HOME}/win-x64/OpenTelemetry.AutoInstrumentation.Native.dll
DOTNET_ADDITIONAL_DEPS	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/AdditionalDeps .
DOTNET_SHARED_STORE	Set this to the folder path of ${OTEL_DOTNET_AUTO_HOME}/store .
DOTNET_STARTUP_HOOKS	Set this to path of the managed assembly ${OTEL_DOTNET_AUTO_HOME}/net/OpenTelemetry.AutoInstrumentation.StartupHook.dll to run before the main application's entry point.

5. Mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure. For Linux, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
        {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "/otel-auto-instrumentation/linux-x64/
OpenTelemetry.AutoInstrumentation.Native.so"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "/otel-auto-instrumentation/AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "/otel-auto-instrumentation/store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "/otel-auto-instrumentation/net/
OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "/otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
```

```
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-
name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://localhost:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
            "name": "OTEL_TRACES_SAMPLER_ARG",
            "value": "endpoint=http://CW_CONTAINER_IP:2000"
```

```
        },
         {
             "name": "OTEL_PROPAGATORS",
             "value": "tracecontext,baggage,b3,xray"
         }
    ],
      "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "/otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
   ]
}
```

For Windows Server, use the following.

```
{
    "name": "my-app",
   ...
    "environment": [
       {
           "name": "OTEL_RESOURCE_ATTRIBUTES",
           "value": "service.name=$SVC_NAME"
       },
        {
            "name": "CORECLR_ENABLE_PROFILING",
            "value": "1"
        },
        {
            "name": "CORECLR_PROFILER",
```

```
            "value": "{918728DD-259F-4A6A-AC2B-B85E1B658318}"
        },
        {
            "name": "CORECLR_PROFILER_PATH",
            "value": "C:\\otel-auto-instrumentation\\win-x64\
\OpenTelemetry.AutoInstrumentation.Native.dll"
        },
        {
            "name": "DOTNET_ADDITIONAL_DEPS",
            "value": "C:\\otel-auto-instrumentation\\AdditionalDeps"
        },
        {
            "name": "DOTNET_SHARED_STORE",
            "value": "C:\\otel-auto-instrumentation\\store"
        },
        {
            "name": "DOTNET_STARTUP_HOOKS",
            "value": "C:\\otel-auto-instrumentation\\net\
\OpenTelemetry.AutoInstrumentation.StartupHook.dll"
        },
        {
            "name": "OTEL_DOTNET_AUTO_HOME",
            "value": "C:\\otel-auto-instrumentation"
        },
        {
            "name": "OTEL_DOTNET_AUTO_PLUGINS",
            "value": "AWS.Distro.OpenTelemetry.AutoInstrumentation.Plugin,
 AWS.Distro.OpenTelemetry.AutoInstrumentation"
        },
        {
            "name": "OTEL_RESOURCE_ATTRIBUTES",
            "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=dotnet-service-
name"
        },
        {
            "name": "OTEL_LOGS_EXPORTER",
            "value": "none"
        },
        {
            "name": "OTEL_METRICS_EXPORTER",
            "value": "none"
        },
        {
```

```
            "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
            "value": "http/protobuf"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
            "value": "true"
        },
        {
            "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316/v1/traces"
        },
        {
            "name": "OTEL_EXPORTER_OTLP_ENDPOINT",
            "value": "http://CW_CONTAINER_IP:4316"
        },
        {
           "name": "OTEL_TRACES_SAMPLER",
           "value": "xray"
       },
       {
           "name": "OTEL_TRACES_SAMPLER_ARG",
           "value": "endpoint=http://CW_CONTAINER_IP:2000"
       },
        {
            "name": "OTEL_PROPAGATORS",
            "value": "tracecontext,baggage,b3,xray"
        }
    ],
    "mountPoints": [
        {
            "sourceVolume": "opentelemetry-auto-instrumentation",
            "containerPath": "C:\\otel-auto-instrumentation",
            "readOnly": false
        }
    ],
    "dependsOn": [
        {
            "containerName": "init",
            "condition": "SUCCESS"
        }
```

```
    ]
}
```

Node.js

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, see Setting up a Node.js application with the ESM module format before you start these steps.

To instrument your application on Amazon ECS with the CloudWatch agent

1. First, specify a bind mount. The volume will be used to share files across containers in the next steps. You will use this bind mount later in this procedure.

```
"volumes": [
  {
    "name": "opentelemetry-auto-instrumentation-node"
  }
]
```

2. Append a new container `init` to your application's task definition. Replace *$IMAGE* with the latest image from the AWS Distro for OpenTelemetry Amazon ECR image repository.

```
{
  "name": "init",
  "image": "$IMAGE",
  "essential": false,
  "command": [
    "cp",
    "-a",
    "/autoinstrumentation/.",
    "/otel-auto-instrumentation-node"
  ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
```

```
  ],
}
```

3. Add a dependency on the `init` container to make sure that this container finishes before your application container starts.

```
"dependsOn": [
  {
    "containerName": "init",
    "condition": "SUCCESS"
  }
]
```

4. Add the following environment variables to your application container.

Environment variable	Setting to enable Application Signals
`OTEL_RESOURCE_ATTRIBUTES`	Specify the following information as key-value pairs: • `service.name` sets the name of the service. This will be diplayed as the service name for your application in Application Signals dashboards. If you don't provide a value for this key, the default of `UnknownService` is used. • `deployment.environment` parameter defines your application's runtime environment and controls the `Environment` dimension for Amazon ECS Application metrics in CloudWatch. When not specified, CloudWatch Agent automatically uses the Amazon ECS Cluster name This attribute key is used only by Applicati on Signals, and is converted into X-Ray

Environment variable	Setting to enable Application Signals
	trace annotations and CloudWatch metric dimensions. (Optional) To enable log correlation for Application Signals, set an additiona l environment variable `aws.log.g roup.names` to be the log group name for your application log. By doing so, the traces and metrics from your applicati on can be correlated with the relevant log entries from the log group. For this variable, replace *`$YOUR_APPLICATION_ LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-gro up-1&log-group-2` . To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.
`OTEL_AWS_APPLICATION_SIGNAL S_ENABLED`	Set to `true` to have your container start sending X-Ray traces and CloudWatch metrics to Application Signals.
`OTEL_METRICS_EXPORTER`	Set to `none` to disable other metrics exporters.
`OTEL_LOGS_EXPORTER`	Set to `none` to disable other logs exporters.

Environment variable	Setting to enable Application Signals
`OTEL_EXPORTER_OTLP_PROTOCOL`	Set to `http/protobuf` to send metrics and traces to Application Signals using OTLP/HTTP and protobuf.
`OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT`	Sends metrics to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/metrics` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/metrics, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Sends traces to the CloudWatch daemon container. • For applications running in `host` mode, set this to `http://localhost:4316/v1/traces` . • For applications running in `bridge` mode or `awsvpc` mode, set this to http://*CW_CONTAINER_IP* :4316/v1/traces, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).

Environment variable	Setting to enable Application Signals
OTEL_TRACES_SAMPLER	Set this to xray to set X-Ray as the traces sampler.
OTEL_TRACES_SAMPLER_ARG	Sets the traces sampler endpoint. • For applications running in host mode, set this to http://localhost:2000 . • For applications running in bridge mode or awsvpc mode, set this to http://*CW_CONTAINER_IP* :2000, where *CW_CONTAINER_IP* is the private IP address of the EC2 container instance. You can retrieve this address from the Instance Metadata Service (IMDS).
OTEL_PROPAGATORS	Set xray as one of the propagators.
NODE_OPTIONS	Set to --require *AWS_ADOT_NODE_INSTRUMENTATION_PATH* . Replace *AWS_ADOT_NODE_INSTRUMENTATION_PATH* with the path where the AWS Distro for OpenTelemetry Node.js auto-instrumentation is stored. For example, /otel-auto-instrumentation-node/autoinstrumentation.js

5. Mount the volume opentelemetry-auto-instrumentation-node that you defined in step 1 of this procedure. If you don't need to enable log correlation with metrics and traces, use the following example for a Node.js application. If you want to enable log correlation, see the next step instead.

   For your Application Container, add a dependency on the init container to make sure that container finishes before your application container starts.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "NODE_OPTIONS",
```

```
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

6. (Optional) To enable log correlation, do the following before you mount the volume. In `OTEL_RESOURCE_ATTRIBUTES`, set an additional environment variable `aws.log.group.names` for the log groups of your application. By doing so, the traces and metrics from your application can be correlated with the relevant log entries from these log groups. For this variable, replace *`$YOUR_APPLICATION_LOG_GROUP`* with the log group names for your application. If you have multiple log groups, you can use an ampersand (&) to separate them as in this example: `aws.log.group.names=log-group-1&log-group-2`. To enable metric to log correlation, setting this current environmental variable is enough. For more information, see Enable metric to log correlation. To enable trace to log correlation, you'll also need to change the logging configuration in your application. For more information, see Enable trace to log correlation.

   The following is an example. Use this example to enable log correlation when you mount the volume `opentelemetry-auto-instrumentation` that you defined in step 1 of this procedure.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
```

```
      "value":
 "aws.log.group.names=$YOUR_APPLICATION_LOG_GROUP,service.name=$SVC_NAME"
    },
    {
      "name": "OTEL_LOGS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_METRICS_EXPORTER",
      "value": "none"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_PROTOCOL",
      "value": "http/protobuf"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_ENABLED",
      "value": "true"
    },
    {
      "name": "OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/metrics"
    },
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://CW_CONTAINER_IP:4316/v1/traces"
    },
    {
      "name": "OTEL_TRACES_SAMPLER",
      "value": "xray"
    },
    {
      "name": "OTEL_TRACES_SAMPLER_ARG",
      "value": "endpoint=http://CW_CONTAINER_IP:2000"
    },
    {
      "name": "NODE_OPTIONS",
      "value": "--require /otel-auto-instrumentation-node/
autoinstrumentation.js"
    }
    ],
  "mountPoints": [
    {
      "sourceVolume": "opentelemetry-auto-instrumentation-node",
```

```
      "containerPath": "/otel-auto-instrumentation-node",
      "readOnly": false
    }
  ],
  "dependsOn": [
    {
      "containerName": "init",
      "condition": "SUCCESS"
    }
  ]
}
```

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

For the ESM module format, using the `init` container to inject the Node.js instrumentation SDK doesn't apply. To enable Application Signals for Node.js with ESM, skip steps 1 and 2 in of the previous procedure, and do the following instead.

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. In steps 4 and 5 in the previous procedure, remove the mounting of the volume `opentelemetry-auto-instrumentation-node`:

```
"mountPoints": [
    {
        "sourceVolume": "opentelemetry-auto-instrumentation-node",
        "containerPath": "/otel-auto-instrumentation-node",
        "readOnly": false
    }
 ]
```

Replace the node options with the following.

```
{
    "name": "NODE_OPTIONS",
    "value": "--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs"
}
```

Step 6: Deploy your application

Create a new revision of your task definition and deploy it to your application cluster. You should see two containers in the newly created task:

- `init`– A required container for initializing Application Signals
- *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

(Optional) Step 7: Monitor your application health

After you have enabled your applications on Amazon ECS, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable Application Signals on Amazon ECS using AWS CDK

To enable Application Signals on Amazon ECS using AWS CDK, do the following.

1. Enable Application Signals for your applications – If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services.

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
  'ApplicationSignalsServiceRole', { }
);
```

The Discovery CloudFormation resource grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Instrument your application with the AWS::ApplicationSignals Construct Library in the AWS CDK. The code snippets in this document are provided in *TypeScript*. For other language-specific alternatives, see Supported programming languages for the AWS CDK.

   - **Enable Application Signals on Amazon ECS with sidecar mode**

     a. Configure `instrumentation` to instrument the application with the AWS Distro for OpenTelemetry (ADOT) SDK Agent. The following is an example of instrumenting a Java application. See InstrumentationVersion for all supported language versions.

     b. Specify `cloudWatchAgentSidecar` to configure the CloudWatch Agent as a sidecar container.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super();
    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });

    const fargateTaskDefinition = new ecs.FargateTaskDefinition(this,
 'SampleAppTaskDefinition', {
```

```
      cpu: 2048,
      memoryLimitMiB: 4096,
    });

    fargateTaskDefinition.addContainer('app', {
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
    });

    new appsignals.ApplicationSignalsIntegration(this,
 'ApplicationSignalsIntegration', {
      taskDefinition: fargateTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.JavaInstrumentationVersion.V2_10_0,
      },
      serviceName: 'sample-app',
      cloudWatchAgentSidecar: {
        containerName: 'ecs-cwagent',
        enableLogging: true,
        essential: false,
        cpu: 256,
        memoryLimitMiB: 512,
      }
    });

    new ecs.FargateService(this, 'MySampleApp', {
      cluster: cluster,
      taskDefinition: fargateTaskDefinition,
      desiredCount: 1,
    });
  }
}
```

- **Enable Application Signals on Amazon ECS with daemon mode**

  > **Note**
  >
  > The daemon deployment strategy is not supported on Amazon ECS Fargate and is only supported on Amazon ECS on Amazon EC2.

  a. Run CloudWatch Agent as a daemon service with HOST network mode.

b. Configure `instrumentation` to instrument the application with the ADOT Python Agent.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });

    // Define Task Definition for CloudWatch agent (Daemon)
    const cwAgentTaskDefinition = new ecs.Ec2TaskDefinition(this,
 'CloudWatchAgentTaskDefinition', {
      networkMode: ecs.NetworkMode.HOST,
    });

    new appsignals.CloudWatchAgentIntegration(this,
 'CloudWatchAgentIntegration', {
      taskDefinition: cwAgentTaskDefinition,
      containerName: 'ecs-cwagent',
      enableLogging: false,
      cpu: 128,
      memoryLimitMiB: 64,
      portMappings: [
        {
          containerPort: 4316,
          hostPort: 4316,
        },
        {
          containerPort: 2000,
          hostPort: 2000,
        },
      ],
    });

    // Create the CloudWatch Agent daemon service
```

```
    new ecs.Ec2Service(this, 'CloudWatchAgentDaemon', {
      cluster,
      taskDefinition: cwAgentTaskDefinition,
      daemon: true,  // Runs one container per EC2 instance
    });

    // Define Task Definition for user application
    const sampleAppTaskDefinition = new ecs.Ec2TaskDefinition(this,
 'SampleAppTaskDefinition', {
      networkMode: ecs.NetworkMode.HOST,
    });

    sampleAppTaskDefinition.addContainer('app', {
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
      cpu: 0,
      memoryLimitMiB: 512,
    });

    // No CloudWatch Agent sidecar is needed as application container
 communicates to CloudWatch Agent daemon through host network
    new appsignals.ApplicationSignalsIntegration(this,
 'ApplicationSignalsIntegration', {
      taskDefinition: sampleAppTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.PythonInstrumentationVersion.V0_8_0
      },
      serviceName: 'sample-app'
    });

    new ecs.Ec2Service(this, 'MySampleApp', {
      cluster,
      taskDefinition: sampleAppTaskDefinition,
      desiredCount: 1,
    });
  }
}
```

- **Enable Application Signals on Amazon ECS with replica mode**

  > **Note**
  >
  > Running CloudWatch Agent service using replica mode requires specific security group configurations to enable communication with other services. For Application

Signals functionality, configure the security group with the minimum inbound rules: Port 2000 (HTTP) and Port 4316 (HTTP). This configuration ensures proper connectivity between the CloudWatch Agent and dependent services.

a. Run CloudWatch Agent as a replica service with service connect.

b. Configure `instrumentation` to instrument the application with the ADOT Python Agent.

c. Override environment variables by configuring `overrideEnvironments` to use service connect endpoints to communicate to the CloudWatch agent server.

```
import { Construct } from 'constructs';
import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import { PrivateDnsNamespace } from 'aws-cdk-lib/aws-servicediscovery';

class MyStack extends cdk.Stack {
  public constructor(scope?: Construct, id?: string, props: cdk.StackProps
 = {}) {
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, 'TestVpc', {});
    const cluster = new ecs.Cluster(this, 'TestCluster', { vpc });
    const dnsNamespace = new PrivateDnsNamespace(this, 'Namespace', {
      vpc,
      name: 'local',
    });
    const securityGroup = new ec2.SecurityGroup(this, 'ECSSG', { vpc });
    securityGroup.addIngressRule(securityGroup, ec2.Port.tcpRange(0,
 65535));

    // Define Task Definition for CloudWatch agent (Replica)
    const cwAgentTaskDefinition = new ecs.FargateTaskDefinition(this,
 'CloudWatchAgentTaskDefinition', {});

    new appsignals.CloudWatchAgentIntegration(this,
 'CloudWatchAgentIntegration', {
      taskDefinition: cwAgentTaskDefinition,
      containerName: 'ecs-cwagent',
```

```
      enableLogging: false,
      cpu: 128,
      memoryLimitMiB: 64,
      portMappings: [
        {
          name: 'cwagent-4316',
          containerPort: 4316,
          hostPort: 4316,
        },
        {
          name: 'cwagent-2000',
          containerPort: 2000,
          hostPort: 2000,
        },
      ],
    });

    // Create the CloudWatch Agent replica service with service connect
    new ecs.FargateService(this, 'CloudWatchAgentService', {
      cluster: cluster,
      taskDefinition: cwAgentTaskDefinition,
      securityGroups: [securityGroup],
      serviceConnectConfiguration: {
        namespace: dnsNamespace.namespaceArn,
        services: [
          {
            portMappingName: 'cwagent-4316',
            dnsName: 'cwagent-4316-http',
            port: 4316,
          },
          {
            portMappingName: 'cwagent-2000',
            dnsName: 'cwagent-2000-http',
            port: 2000,
          },
        ],
      },
      desiredCount: 1,
    });

    // Define Task Definition for user application
    const sampleAppTaskDefinition = new ecs.FargateTaskDefinition(this,
'SampleAppTaskDefinition', {});
```

```
    sampleAppTaskDefinition.addContainer('app', {
      image: ecs.ContainerImage.fromRegistry('test/sample-app'),
      cpu: 0,
      memoryLimitMiB: 512,
    });

    // Overwrite environment variables to connect to the CloudWatch Agent
service just created
    new appsignals.ApplicationSignalsIntegration(this,
'ApplicationSignalsIntegration', {
      taskDefinition: sampleAppTaskDefinition,
      instrumentation: {
        sdkVersion: appsignals.PythonInstrumentationVersion.V0_8_0,
      },
      serviceName: 'sample-app',
      overrideEnvironments: [
        {
          name:
appsignals.CommonExporting.OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT,
          value: 'http://cwagent-4316-http:4316/v1/metrics',
        },
        {
          name:
appsignals.TraceExporting.OTEL_EXPORTER_OTLP_TRACES_ENDPOINT,
          value: 'http://cwagent-4316-http:4316/v1/traces',
        },
        {
          name: appsignals.TraceExporting.OTEL_TRACES_SAMPLER_ARG,
          value: 'endpoint=http://cwagent-2000-http:2000',
        },
      ],
    });

    // Create ECS Service with service connect configuration
    new ecs.FargateService(this, 'MySampleApp', {
      cluster: cluster,
      taskDefinition: sampleAppTaskDefinition,
      serviceConnectConfiguration: {
        namespace: dnsNamespace.namespaceArn,
      },
      desiredCount: 1,
    });
  }
```

```
}
```

3. Setting up a Node.js application with the ESM module format. There is limited support for Node.js applications with the ESM module format. For more information, see Known limitations about Node.js with ESM.

   For the ESM module format, enabling Application Signals by using the `init` container to inject the Node.js instrumentation SDK doesn't apply. Skip Step 2 in this procedure, and do the following instead.

   - Install the relevant dependencies to your Node.js application for autoinstrumentation.

     ```
     npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
     npm install @opentelemetry/instrumentation@0.54.
     ```

   - Update TaskDefinition.

     a. Add additional configuration to your application container.

     b. Configure `NODE_OPTIONS`.

     c. (Optional) Add CloudWatch Agent if you choose sidecar mode.

     ```
     import { Construct } from 'constructs';
     import * as appsignals from '@aws-cdk/aws-applicationsignals-alpha';
     import * as ecs from 'aws-cdk-lib/aws-ecs';

     class MyStack extends cdk.Stack {
       public constructor(scope?: Construct, id?: string, props: cdk.StackProps
      = {}) {
         super(scope, id, props);
         const fargateTaskDefinition = new ecs.FargateTaskDefinition(stack,
      'TestTaskDefinition', {
           cpu: 256,
           memoryLimitMiB: 512,
         });
         const appContainer = fargateTaskDefinition.addContainer('app', {
           image: ecs.ContainerImage.fromRegistry('docker/cdk-test'),
         });

         const volumeName = 'opentelemetry-auto-instrumentation'
         fargateTaskDefinition.addVolume({name: volumeName});

         // Inject additional configurations
     ```

```
    const injector = new appsignals.NodeInjector(volumeName,
 appsignals.NodeInstrumentationVersion.V0_5_0);
    injector.renderDefaultContainer(fargateTaskDefinition);
    // Configure NODE_OPTIONS
    appContainer.addEnvironment('NODE_OPTIONS', '--import @aws/aws-
distro-opentelemetry-node-autoinstrumentation/register --experimental-
loader=@opentelemetry/instrumentation/hook.mjs')

    // Optional: add CloudWatch agent
    const cwAgent = new appsignals.CloudWatchAgentIntegration(stack,
 'AddCloudWatchAgent', {
      containerName: 'ecs-cwagent',
      taskDefinition: fargateTaskDefinition,
      essential: false,
      memoryReservationMiB: 50,
    });
    appContainer.addContainerDependencies({
      container: cwAgent.agentContainer,
      condition: ecs.ContainerDependencyCondition.START,
    });
}
```

4. Deploy the updated stack – Run the `cdk synth` command in your application's main directory. To deploy the service in your AWS account, run the `cdk deploy` command in your application's main directory.

   If you used the sidecar strategy, you'll see one service created:

   - *`APPLICATION_SERVICE`* is the service of your application. It includes the three following containers:
     - `init`– A required container for initializing Application Signals.
     - `ecs-cwagent`– A container running the CloudWatch agent
     - *`my-app`*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

   If you used the daemon strategy, you'll see two services created:

   - *`CloudWatchAgentDaemon`* is the CloudWatch agent daemon service.

- *APPLICATION_SERVICE* is the service of your application. It includes the two following containers:
  - `init`– A required container for initializing Application Signals.
  - *my-app*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

If you used the replica strategy, you'll see two services created:

- *CloudWatchAgentService* is the CloudWatch agent replica service.
- *APPLICATION_SERVICE* is the service of your application. It includes the two following containers:
  - `init`– A required container for initializing Application Signals.
  - *my-app*– This is the example application container in our documentation. In your actual workloads, this specific container might not exist or might be replaced with your own service containers.

Enable Application Signals on Amazon ECS using Model Context Protocol (MCP)

You can use the CloudWatch Application Signals Model Context Protocol (MCP) server to enable Application Signals on your Amazon ECS clusters through conversational AI interactions. This provides a natural language interface for setting up Application Signals monitoring.

The MCP server automates the enablement process by understanding your requirements and generating the appropriate configuration. Instead of manually following console steps or writing CDK code, you can simply describe what you want to enable.

Prerequisites

Before using the MCP server to enable Application Signals, make sure you have:

- A Development Environment that supports MCP (such as Kiro, Claude Desktop, VSCode with MCP extensions, or other MCP-compatible tools)
- The CloudWatch Application Signals MCP server configured in your IDE. For detailed setup instructions, see CloudWatch Application Signals MCP Server documentation.

Using the MCP server

After you have configured the CloudWatch Application Signals MCP server in your IDE, you can request enablement guidance using natural language prompts. While the coding assistant can infer context from your project structure, providing specific details in your prompts helps make sure that you receive more accurate and relevant guidance. Include information such as your application language, Amazon ECS cluster name, deployment strategy (sidecar or daemon), and absolute paths to your infrastructure and application code.

Best practice prompts (specific and complete):

```
"Enable Application Signals for my Python service running on ECS.
My app code is in /home/user/flask-api and IaC is in /home/user/flask-api/terraform"

"I want to add observability to my Node.js application on ECS cluster 'production-
cluster' using sidecar deployment.
The application code is at /Users/dev/checkout-service and
the task definitions are at /Users/dev/checkout-service/ecs"

"Help me instrument my Java Spring Boot application on ECS with Application Signals
 using daemon strategy.
Application directory: /opt/apps/payment-api
CDK infrastructure: /opt/apps/payment-api/cdk"
```

Less effective prompts:

```
"Enable monitoring for my app"
# Missing: platform, language, paths

"Enable Application Signals. My code is in ./src and IaC is in ./infrastructure"
# Problem: Relative paths instead of absolute paths

"Enable Application Signals for my ECS service at /home/user/myapp"
# Missing: programming language, deployment strategy
```

Quick template:

```
"Enable Application Signals for my [LANGUAGE] service on ECS.
Deployment strategy: [sidecar/daemon]
App code: [ABSOLUTE_PATH_TO_APP]
IaC code: [ABSOLUTE_PATH_TO_IAC]"
```

Benefits of using the MCP server

Using the CloudWatch Application Signals MCP server offers several advantages:

- **Natural language interface:** Describe what you want to enable without memorizing commands or configuration syntax
- **Context-aware guidance:** The MCP server understands your specific environment and provides tailored recommendations
- **Reduced errors:** Automated configuration generation minimizes manual typing errors
- **Faster setup:** Get from intention to implementation more quickly
- **Learning tool:** See the generated configurations and understand how Application Signals works

Additional resources

For more information about configuring and using the CloudWatch Application Signals MCP server, see the MCP server documentation.

Enable your applications on Kubernetes

Enable CloudWatch Application Signals on Kubernetes by using the custom setup steps described in this section.

For applications running on Kubernetes, you install and configure the CloudWatch agent and AWS Distro for OpenTelemetry yourself. On these architectures enabled with a custom Application Signals setup, Application Signals doesn't autodiscover the names of your services or the hosts or clusters they run on. You must specify these names during the custom setup, and the names that you specify are what is displayed on Application Signals dashboards.

Requirements

- You have admininstrator permission on the Kubernetes cluster where you are enabling Application Signals.
- You must have the AWS CLI installed on the environment where your Kubernetes cluster is running. For more information about installing the AWS CLI, see Install or update the latest version of the AWS CLI.
- You have kubectl and helm installed on your local terminal. For more information, see the kubectl and Helm documentation.

Step 1: Enable Application Signals in your account

You must first enable Application Signals in your account. If you haven't, see Enable Application Signals in your account.

Step 2: Install the CloudWatch agent operator in your cluster

Installing the CloudWatch agent operator installs the operator, the CloudWatch agent, and other auto-instrumentation into your cluster. To do so, enter the following command. Replace *$REGION* with your AWS Region. Replace *$YOUR_CLUSTER_NAME* with the name that you want to appear for your cluster in Application Signals dashboards.

```
helm repo add aws-observability https://aws-observability.github.io/helm-charts
helm install amazon-cloudwatch-operator aws-observability/amazon-cloudwatch-
observability \
--namespace amazon-cloudwatch --create-namespace \
--set region=$REGION \
--set clusterName=$YOUR_CLUSTER_NAME
```

For more information, see amazon-cloudwatch-observability on GitHub.

Step 3: Set up AWS credentials for your Kubernetes clusters

> **Important**
>
> If your Kubernetes cluster is hosted on Amazon EC2, you can skip this section and proceed to Step 4: Add annotations.

If your Kubernetes cluster is hosted on-premises, you must use the instructions in this section to add AWS credentials to your Kubernetes environment.

To set up permissions for an on-premises Kubernetes cluster

1. Create the IAM user to be used to provide permissions to your on-premises host:
   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. Choose **Users**, **Create User**.
   c. In **User details**, for **User name**, enter a name for the new IAM user. This is the sign-in name for AWS that will be used to authenticate your host. Then choose **Next**

d. On the **Set permissions** page, under **Permissions options**, select **Attach policies directly**.

e. From the **Permissions policies** list, select the **CloudWatchAgentServerPolicy** policy to add to your user. Then choose **Next**.

f. On the **Review and create** page, make sure that you are satisfied with the user name and that the **CloudWatchAgentServerPolicy** policy is in the **Permissions summary**.

g. Choose **Create user**

2. Create and retrieve your AWS access key and secret key:

a. In the navigation pane in the IAM console, choose **Users** and then select the user name of the user that you created in the previous step.

b. On the user's page, choose the **Security credentials** tab. Then, in the **Access keys** section, choose **Create access key**.

c. For **Create access key Step 1**, choose **Command Line Interface (CLI)**.

d. For **Create access key Step 2**, optionally enter a tag and then choose **Next**.

e. For **Create access key Step 3**, select **Download .csv file** to save a .csv file with your IAM user's access key and secret access key. You need this information for the next steps.

f. Choose **Done**.

3. Configure your AWS credentials in your on-premises host by entering the following command. Replace *ACCESS_KEY_ID* and *SECRET_ACCESS_ID* with your newly generated access key and secret access key from the .csv file that you downloaded in the previous step. By default, the credential file is saved in `/home/user/.aws/credentials`.

```
$ aws configure --profile AmazonCloudWatchAgent
AWS Access Key ID [None]: ACCESS_KEY_ID
AWS Secret Access Key [None]: SECRET_ACCESS_ID
Default region name [None]: MY_REGION
Default output format [None]: json
```

4. Edit the custom resource that the CloudWatch agent installed using the Helm chart to add the newly created AWS credentials secret.

```
kubectl edit amazoncloudwatchagent cloudwatch-agent -n amazon-cloudwatch
```

5. While your file editor is open mount the AWS credentials into the CloudWatch agent container by adding the following configuration to the top of the deployment. Replace the path `/home/user/.aws/credentials` with the location of your local AWS credentials file.

```
apiVersion: cloudwatch.aws.amazon.com/v1alpha1
kind: AmazonCloudWatchAgent
metadata:
  name: cloudwatch-agent
  namespace: amazon-cloudwatch
spec:
  volumeMounts:
  - mountPath: /rootfs
    volumeMounts:
    - name: aws-credentials
      mountPath: /root/.aws
      readOnly: true
    volumes:
    - hostPath:
      path: /home/user/.aws/credentials
    name: aws-credentials
---
```

Step 4: Add annotations

> **Note**
>
> If you are enabling Application Signals for a Node.js application with ESM, skip the steps in this section and see the section called "Setting up a Node.js application with the ESM module format" instead.

The next step is to instrument your application for CloudWatch Application Signals by adding a language-specific annotation to your Kubernetes workload or namespace. This annotation auto-instruments your application to send metrics, traces, and logs to Application Signals.

To add the annotations for Application Signals

1. You have two options for the annotation:

   - **Annotate Workload** auto-instruments a single workload in a cluster.
   - **Annotate Namespace** auto-instruments all workloads deployed in the selected namespace.

Choose one of those options, and follow the appropriate steps.

2. To annotate a single workload, enter one of the following commands. Replace *$WORKLOAD_TYPE* and *$WORKLOAD_NAME* with the values for your workload.

   - For Java workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-java":
      "true"}}}}}'
     ```

   - For Python workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-python":
      "true"}}}}}'
     ```

     For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.

   - For .NET workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-dotnet":
      "true"}}}}}'
     ```

     Note

     To enable Application Signals for a .NET workload on Alpine Linux (`linux-musl-x64`) based images, add the following additional annotation.

     ```
     instrumentation.opentelemetry.io/otel-dotnet-auto-runtime: "linux-musl-
     x64"
     ```

   - For Node.js workloads:

     ```
     kubectl patch $WORKLOAD_TYPE $WORKLOAD_NAME -p '{"spec": {"template":
      {"metadata": {"annotations": {"instrumentation.opentelemetry.io/inject-nodejs":
      "true"}}}}}'
     ```

3. To annotate all workloads in a namespace, enter enter one of the following commands. Replace *$NAMESPACE* with the name of your namespace.

   If the namespace includes Java, Python, and .NET workloads, add all annotations to the namespace.

   - For Java workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-java=true
     ```

   - For Python workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     python=true
     ```

     For Python applications, there are additional required configurations. For more information, see Python application doesn't start after Application Signals is enabled.

   - For .NET workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     dotnet=true
     ```

   - For Node.js workloads in the namespace:

     ```
     kubectl annotate ns $NAMESPACE instrumentation.opentelemetry.io/inject-
     nodejs=true
     ```

   After adding the annotations, restart all pods in the namespace by entering the following command:

   ```
   kubectl rollout restart
   ```

4. When the previous steps are completed, in the CloudWatch console, choose **Application Signals**, **Services**. This opens the dashboards where you can see the data that Application Signals collects. It might take a few minutes for data to appear.

   For more information about the **Services** view, see Monitor the operational health of your applications with Application Signals.

Setting up a Node.js application with the ESM module format

We provide limited support for Node.js applications with the ESM module format. For details, see the section called “Known limitations about Node.js with ESM”.

For the ESM module format, enabling Application Signals by annotating the manifest file doesn't work. Skip the previous procedure and do the following instead:

To enable Application Signals for a Node.js application with ESM

1. Install the relevant dependencies to your Node.js application for autoinstrumentation:

```
npm install @aws/aws-distro-opentelemetry-node-autoinstrumentation
npm install @opentelemetry/instrumentation@0.54.0
```

2. Add the following environmental variables to the Dockerfile for your application and build the image.

```
...
ENV OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true
ENV OTEL_TRACES_SAMPLER_ARG='endpoint=http://cloudwatch-agent.amazon-
cloudwatch:2000'
ENV OTEL_TRACES_SAMPLER='xray'
ENV OTEL_EXPORTER_OTLP_PROTOCOL='http/protobuf'
ENV OTEL_EXPORTER_OTLP_TRACES_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/traces'
ENV OTEL_AWS_APPLICATION_SIGNALS_EXPORTER_ENDPOINT='http://cloudwatch-agent.amazon-
cloudwatch:4316/v1/metrics'
ENV OTEL_METRICS_EXPORTER='none'
ENV OTEL_LOGS_EXPORTER='none'
ENV NODE_OPTIONS='--import @aws/aws-distro-opentelemetry-node-autoinstrumentation/
register --experimental-loader=@opentelemetry/instrumentation/hook.mjs'
ENV OTEL_SERVICE_NAME='YOUR_SERVICE_NAME' #replace with a proper service name
ENV OTEL_PROPAGATORS='tracecontext,baggage,b3,xray'
...

# command to start the application
# for example
# CMD ["node", "index.mjs"]
```

3. Add the environmental variables `OTEL_RESOURCE_ATTRIBUTES_POD_NAME`, `OTEL_RESOURCE_ATTRIBUTES_NODE_NAME`,

OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME, POD_NAMESPACE and OTEL_RESOURCE_ATTRIBUTES to the deployment yaml file for the application. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodejs-app
  labels:
    app: nodejs-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodejs-app
  template:
    metadata:
      labels:
        app: nodejs-app
      # annotations:
      # make sure this annotation doesn't exit
      #   instrumentation.opentelemetry.io/inject-nodejs: 'true'
    spec:
      containers:
      - name: nodejs-app
        image:your-nodejs-application-image #replace it with a proper image uri
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
        env:
          - name: OTEL_RESOURCE_ATTRIBUTES_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: OTEL_RESOURCE_ATTRIBUTES_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.labels['app'] # Assuming 'app' label is set to
 the deployment name
          - name: POD_NAMESPACE
```

```
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: OTEL_RESOURCE_ATTRIBUTES
            value: "k8s.deployment.name=
$(OTEL_RESOURCE_ATTRIBUTES_DEPLOYMENT_NAME),k8s.namespace.name=
$(POD_NAMESPACE),k8s.node.name=$(OTEL_RESOURCE_ATTRIBUTES_NODE_NAME),k8s.pod.name=
$(OTEL_RESOURCE_ATTRIBUTES_POD_NAME)"
```

4. Deploy the Node.js application to the Kubernetes cluster.

(Optional) Step 5: Monitor your application health

After you have enabled your applications on Kubernetes, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Enable your applications on Lambda

You can enable Application Signals for your Lambda functions. Application Signals automatically instruments your Lambda functions using enhanced AWS Distro for OpenTelemetry (ADOT) libraries, provided through a Lambda layer. This AWS Lambda Layer for OpenTelemetry packages and deploys the libraries that are required for auto-instrumentation for Application Signals.

In addition to supporting Application Signals, this Lambda layer is also a component of Lambda OpenTelemetry support and provides tracing functionality.

You can also enhance Lambda observability by using transaction search, which enables the capture of trace spans for Lambda function invocation without sampling. This feature allows you to collect spans for your functions, unaffected by the `sampled` flag in trace context propagation. This ensures that there is no additional impact to downstream dependent services. By enabling transaction search on Lambda, you gain complete visibility into your function performance and you can troubleshoot rarely occurring issues. To get started, see Transaction Search

Topics

- Getting started
- Use the CloudWatch Application Signals console
- Use the Lambda console
- Enable Application Signals on Lambda using AWS CDK

- Enable Application Signals on Lambda using Model Context Protocol (MCP)
- (Optional) Monitor your application health
- Manually enable Application Signals.
- Manually disable Application Signals
- Configuring Application Signals
- AWS Lambda Layer for OpenTelemetry ARNs
- Deploy Lambda functions using Amazon ECR container

Getting started

There are three methods for enabling Application Signals for your Lambda functions.

After you enable Application Signals for a Lambda function, it takes a few minutes for telemetry from that function to appear in the Application Signals console.

- Use the CloudWatch Application Signals console
- Use the Lambda console
- Manually add the AWS Lambda Layer for OpenTelemetry to your Lambda function runtime.

Each of these methods adds the AWS Lambda Layer for OpenTelemetry to your function.

Use the CloudWatch Application Signals console

Use these steps to use the Application Signals console to enable Application Signals for a Lambda function.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Services**.
3. In the **Services** list area, choose **Enable Application Signals**.
4. Choose the **Lambda** tile.
5. Select each function that you want to enable for Application Signals, and then choose **Done**.

Use the Lambda console

Use these steps to use the Lambda console to enable Application Signals for a Lambda function.

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. In the navigation pane, choose **Functions** and then choose the name of the function that you want to enable.
3. Choose the **Configuration** tab, and then choose **Monitoring and operations tools**.
4. Choose **Edit**.
5. In the **CloudWatch Application Signals and X-Ray** section, select both **Automatically collect application traces and standard application metrics with Application Signals** and **Automatically collect Lambda service traces for end to end visibility with X-Ray.**.
6. Choose **Save**.

Enable Application Signals on Lambda using AWS CDK

If you haven't enabled Application Signals in this account yet, you must grant Application Signals the permissions it needs to discover your services. For more information, see Enable Application Signals in your account.

1. Enable Application Signals for your applications

```
import { aws_applicationsignals as applicationsignals } from 'aws-cdk-lib';

const cfnDiscovery = new applicationsignals.CfnDiscovery(this,
  'ApplicationSignalsServiceRole', { }
);
```

The Discovery CloudFormation resource grants Application Signals the following permissions:

- `xray:GetServiceGraph`
- `logs:StartQuery`
- `logs:GetQueryResults`
- `cloudwatch:GetMetricData`
- `cloudwatch:ListMetrics`
- `tag:GetResources`

For more information about this role, see Service-linked role permissions for CloudWatch Application Signals.

2. Add the IAM policy `CloudWatchLambdaApplicationSignalsExecutionRolePolicy` to the lambda function.

```
const fn = new Function(this, 'DemoFunction', {
    code: Code.fromAsset('$YOUR_LAMBDA.zip'),
    runtime: Runtime.PYTHON_3_12,
    handler: '$YOUR_HANDLER'
})

fn.role?.addManagedPolicy(ManagedPolicy.fromAwsManagedPolicyName('CloudWatchLambdaApplicati
```

3. Replace `$AWS_LAMBDA_LAYER_FOR_OTEL_ARN` with the actual AWS Lambda Layer for OpenTelemetry ARN for your Region.

```
fn.addLayers(LayerVersion.fromLayerVersionArn(
    this, 'AwsLambdaLayerForOtel',
    '$AWS_LAMBDA_LAYER_FOR_OTEL_ARN'
))
fn.addEnvironment("AWS_LAMBDA_EXEC_WRAPPER", "/opt/otel-instrument");
```

Enable Application Signals on Lambda using Model Context Protocol (MCP)

You can use the CloudWatch Application Signals Model Context Protocol (MCP) server to enable Application Signals on your Lambda functions through conversational AI interactions. This provides a natural language interface for setting up Application Signals monitoring.

The MCP server automates the enablement process by understanding your requirements and generating the appropriate configuration. Instead of manually following console steps or writing CDK code, you can simply describe what you want to enable.

Prerequisites

Before using the MCP server to enable Application Signals, make sure you have:

- A Development Environment that supports MCP (such as Kiro, Claude Desktop, VSCode with MCP extensions, or other MCP-compatible tools)
- The CloudWatch Application Signals MCP server configured in your IDE. For detailed setup instructions, see CloudWatch Application Signals MCP Server documentation.

Using the MCP server

After you have configured the CloudWatch Application Signals MCP server in your IDE, you can request enablement guidance using natural language prompts. While the coding assistant can infer context from your project structure, providing specific details in your prompts helps make sure that you receive more accurate and relevant guidance. Include information such as your Lambda function's programming language, function name, and absolute paths to your Lambda function code and infrastructure code.

Best practice prompts (specific and complete):

```
"Enable Application Signals for my Python Lambda function.
My function code is in /home/user/order-processor/lambda and IaC is in /home/user/
order-processor/terraform"

"I want to add observability to my Node.js Lambda function 'checkout-handler'.
The function code is at /Users/dev/checkout-function and
the CDK infrastructure is at /Users/dev/checkout-function/cdk"

"Help me instrument my Java Lambda function with Application Signals.
Function directory: /opt/apps/payment-lambda
CDK infrastructure: /opt/apps/payment-lambda/cdk"
```

Less effective prompts:

```
"Enable monitoring for my Lambda"
# Missing: language, paths

"Enable Application Signals. My code is in ./src and IaC is in ./infrastructure"
# Problem: Relative paths instead of absolute paths

"Enable Application Signals for my Lambda at /home/user/myfunction"
# Missing: programming language
```

Quick template:

```
"Enable Application Signals for my [LANGUAGE] Lambda function.
Function code: [ABSOLUTE_PATH_TO_FUNCTION]
IaC code: [ABSOLUTE_PATH_TO_IAC]"
```

Benefits of using the MCP server

Using the CloudWatch Application Signals MCP server offers several advantages:

- **Natural language interface:** Describe what you want to enable without memorizing commands or configuration syntax
- **Context-aware guidance:** The MCP server understands your specific environment and provides tailored recommendations
- **Reduced errors:** Automated configuration generation minimizes manual typing errors
- **Faster setup:** Get from intention to implementation more quickly
- **Learning tool:** See the generated configurations and understand how Application Signals works

For more information about configuring and using the CloudWatch Application Signals MCP server, see the MCP server documentation.

(Optional) Monitor your application health

After you have enabled your applications on Lambda, you can monitor your application health. For more information, see Monitor the operational health of your applications with Application Signals.

Manually enable Application Signals.

Use these steps to manually enable Application Signals for a Lambda function.

1. Add the AWS Lambda Layer for OpenTelemetry to your Lambda runtime. To find the layer ARN for your Region, see ADOT Lambda Layer ARNs.
2. Add the environment variable `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument`

   Add the environment variable `LAMBDA_APPLICATION_SIGNALS_REMOTE_ENVIRONMENT` to configure custom Lambda environments. By default, lambda environments are configured to `lambda:default`.
3. Attach the AWS managed IAM policy **CloudWatchLambdaApplicationSignalsExecutionRolePolicy** to the Lambda execution role.
4. (Optional) We recommend that you enable Lambda active tracing to get a better tracing experience. For more information, see Visualize Lambda function invocations using AWS X-Ray.

Manually disable Application Signals

To manually disable Application Signals for a Lambda function, remove the AWS Lambda Layer for OpenTelemetry from your Lambda runtime, and remove the `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument` environment variable.

Configuring Application Signals

You can use this section to configure Application Signals in Lambda.

Grouping multiple Lambda functions into one service

Environment variable `OTEL_SERVICE_NAME` sets the name of the service. This will be displayed as the service name for your application in Application Signals dashboards. You can assign the same service name to multiple Lambda functions, and they will be merged into a single service in Application Signals. When you don't provide a value for this key, the default Lambda Function name is used.

Sampling

By default, the trace sampling strategy is parent based. You can adjust the sampling strategy by setting environment variables `OTEL_TRACES_SAMPLER`.

For example, set trace sampling rate to 30%.

```
OTEL_TRACES_SAMPLER=traceidratio
OTEL_TRACES_SAMPLER_ARG=0.3
```

For more information , see OpenTelemetry Environment Variable Specification.

Enabling all library instrumentation's

To reduce Lambda cold starts, by default, only AWS SDK and HTTP instrumentation's are enabled for Python, Node, and Java. You can set environment variables to enable instrumentation for other libraries used in your Lambda function.

- Python – `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=none`
- Node – `OTEL_NODE_DISABLED_INSTRUMENTATIONS=none`
- Java – `OTEL_INSTRUMENTATION_COMMON_DEFAULT_ENABLED=true`

AWS Lambda Layer for OpenTelemetry ARNs

For the complete list of AWS Lambda Layer for OpenTelemetry ARNs by Region and runtime, see ADOT Lambda Layer ARNs in the AWS Distro for OpenTelemetry documentation. The layer is available for Python, Node.js, .NET, and Java runtimes.

Deploy Lambda functions using Amazon ECR container

Lambda functions deployed as container images do not support Lambda Layers in the traditional way. When using container images, you cannot attach a layer as you would with other Lambda deployment methods. Instead, you must manually incorporate the layer's contents into your container image during the build process.

Java

You can learn how to integrate the AWS Lambda Layer for OpenTelemetry into your containerized Java Lambda function, download the `layer.zip` artifact, and integrate it into your Java Lambda function container to enable Application Signals monitoring.

Prerequisites

- AWS CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function

   ```
   mkdir java-appsignals-container-lambda && \
   cd java-appsignals-container-lambda
   ```

   Create a Maven project structure

   ```
   mkdir -p src/main/java/com/example/java/lambda
   mkdir -p src/main/resources
   ```

2. **Create Dockerfile**

Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

```
FROM public.ecr.aws/lambda/java:21

# Install utilities
RUN dnf install -y unzip wget maven

# Download the OpenTelemetry Layer with AppSignals Support
RUN wget https://github.com/aws-observability/aws-otel-java-instrumentation/
releases/latest/download/layer.zip -O /tmp/layer.zip

# Extract and include Lambda layer contents
RUN mkdir -p /opt && \
    unzip /tmp/layer.zip -d /opt/ && \
    chmod -R 755 /opt/ && \
    rm /tmp/layer.zip

# Copy and build function code
COPY pom.xml ${LAMBDA_TASK_ROOT}
COPY src ${LAMBDA_TASK_ROOT}/src
RUN mvn clean package -DskipTests

# Copy the JAR file to the Lambda runtime directory (from inside the container)
RUN mkdir -p ${LAMBDA_TASK_ROOT}/lib/
RUN cp ${LAMBDA_TASK_ROOT}/target/function.jar ${LAMBDA_TASK_ROOT}/lib/

# Set the handler
CMD ["com.example.java.lambda.App::handleRequest"]
```

Note

The `layer.zip` file contains the OpenTelemetry instrumentation necessary for AWS Application Signals support to monitor your Lambda function.
The layer extraction steps ensures:

- The layer.zip contents are properly extracted to the `/opt/ directory`
- The `otel-instrument` script receives proper execution permissions
- The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code** – Create a Java file for your Lambda handler at `src/main/java/com/example/lambda/App.java`:

   Your project should look something like:

```
.
### Dockerfile
### pom.xml
### src
    ### main
        ### java
        #   ### com
        #       ### example
        #           ### java
        #               ### lambda
        #                   ### App.java
        ### resources
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with ECR

   First with public ECR (for base image):

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Then with your private ECR:

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

Build, tag and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest
```

5. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

.Net

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized .Net Lambda function, download the `layer.zip` artifact, and integrate it into your .Net Lambda function to enable Application Signals monitoring.

Prerequisites

- AWS CLI configured with your credentials
- Docker installed
- .Net 8 SDK
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

   ```
   mkdir dotnet-appsignals-container-lambda && \
   cd dotnet-appsignals-container-lambda
   ```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

   ```
   FROM public.ecr.aws/lambda/dotnet:8

   # Install utilities
   RUN dnf install -y unzip wget dotnet-sdk-8.0 which

   # Add dotnet command to docker container's PATH
   ENV PATH="/usr/lib64/dotnet:${PATH}"

   # Download the OpenTelemetry Layer with AppSignals Support
   RUN wget https://github.com/aws-observability/aws-otel-dotnet-instrumentation/
   releases/latest/download/layer.zip -O /tmp/layer.zip

   # Extract and include Lambda layer contents
   RUN mkdir -p /opt && \
       unzip /tmp/layer.zip -d /opt/ && \
       chmod -R 755 /opt/ && \
   ```

```
    rm /tmp/layer.zip

WORKDIR ${LAMBDA_TASK_ROOT}

# Copy the project files
COPY dotnet-lambda-function/src/dotnet-lambda-function/*.csproj
 ${LAMBDA_TASK_ROOT}/
COPY dotnet-lambda-function/src/dotnet-lambda-function/Function.cs
 ${LAMBDA_TASK_ROOT}/
COPY dotnet-lambda-function/src/dotnet-lambda-function/aws-lambda-tools-
defaults.json ${LAMBDA_TASK_ROOT}/

# Install dependencies and build the application
RUN dotnet restore

# Use specific runtime identifier and disable ReadyToRun optimization
RUN dotnet publish -c Release -o out --self-contained false /
p:PublishReadyToRun=false

# Copy the published files to the Lambda runtime directory
RUN cp -r out/* ${LAMBDA_TASK_ROOT}/

CMD ["dotnet-lambda-function::dotnet_lambda_function.Function::FunctionHandler"]
```

Note

The `layer.zip` file contains the OpenTelemetry instrumentation necessary for AWS Application Signals support to monitor your Lambda function.
The layer extraction steps ensures:

- The layer.zip contents are properly extracted to the `/opt/ directory`
- The `otel-instrument` script receives proper execution permissions
- The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code** – Initialize your Lambda project using the AWS Lambda .NET template:

```
# Install the Lambda templates if you haven't already
dotnet new -i Amazon.Lambda.Templates

# Create a new Lambda project
```

```
dotnet new lambda.EmptyFunction -n dotnet-lambda-function
```

Your project should look something like:

```
.
### Dockerfile
### dotnet-lambda-function
    ### src
    #   ### dotnet-lambda-function
    #       ### Function.cs
    #       ### Readme.md
    #       ### aws-lambda-tools-defaults.json
    #       ### dotnet-lambda-function.csproj
    ### test
        ### dotnet-lambda-function.Tests
            ### FunctionTest.cs
            ### dotnet-lambda-function.Tests.csproj
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output text)
# set AWS_REGION (aws configure get region)
```

   Update the `Function.cs` code to:

   Update the `dotnet-lambda-function.csproj` code to:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0>/TargetFramework>
   <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>

    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
```

```
    <PublishReadyToRun>true</PublishReadyToRun>
 </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.5.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
 Version="2.4.4" />
    <PackageReference Include="AWSSDK.S3" Version="3.7.305.23" />
  </ItemGroup>
</Project>
```

5. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

   Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

   Build, tag, and push your image

```
# Build the Docker image
```

```
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest

5. Create and Configure the Lambda Function
```

6. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

7. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Node.js

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized Node.js Lambda function, download the `layer.zip` artifact, and integrate it into your Node.js Lambda function to enable Application Signals monitoring.

Prerequisites

- AWS CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

   ```
   mkdir nodejs-appsignals-container-lambda &&\
   cd nodejs-appsignals-container-lambda
   ```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

   ```
   # Dockerfile
   FROM public.ecr.aws/lambda/nodejs:22

   # Install utilities
   RUN dnf install -y unzip wget

   # Download the OpenTelemetry Layer with AppSignals Support
   RUN wget https://github.com/aws-observability/aws-otel-js-instrumentation/
   releases/latest/download/layer.zip -O /tmp/layer.zip

   # Extract and include Lambda layer contents
   RUN mkdir -p /opt && \
       unzip /tmp/layer.zip -d /opt/ && \
       chmod -R 755 /opt/ && \
       rm /tmp/layer.zip

   # Install npm dependencies
   ```

```
RUN npm init -y
RUN npm install

# Copy function code
COPY *.js ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler
CMD [ "index.handler" ]
```

> **Note**
>
> The `layer.zip` file contains the OpenTelemetry instrumentation necessary for AWS Application Signals support to monitor your Lambda function.
> The layer extraction steps ensures:
>
> - The layer.zip contents are properly extracted to the `/opt/ directory`
> - The `otel-instrument` script receives proper execution permissions
> - The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code**

Create an `index.js` file with the following content:

```
const { S3Client, ListBucketsCommand } = require('@aws-sdk/client-s3');

// Initialize S3 client
const s3Client = new S3Client({ region: process.env.AWS_REGION });

exports.handler = async function(event, context) {
  console.log('Received event:', JSON.stringify(event, null, 2));
  console.log('Handler initializing:', exports.handler.name);

  const response = {
    statusCode: 200,
    body: {}
  };

  try {
    // List S3 buckets
    const command = new ListBucketsCommand({});
    const data = await s3Client.send(command);
```

```
    // Extract bucket names
    const bucketNames = data.Buckets.map(bucket => bucket.Name);

    response.body = {
      message: 'Successfully retrieved buckets',
      buckets: bucketNames
    };

  } catch (error) {
    console.error('Error listing buckets:', error);

    response.statusCode = 500;
    response.body = {
      message: `Error listing buckets: ${error.message}`
    };
  }

  return response;
};
```

Your project structure should look something like this:

```
.
### Dockerfile
### index.js
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

Build, tag, and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest

5. Create and Configure the Lambda Function
```

5. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Python

You can learn how to integrate the OpenTelemetry Layer with Application Signals support into your containerized Python Lambda function, download the `layer.zip` artifact, and integrate it into your Python Lambda function to enable Application Signals monitoring.

Prerequisites

- AWS CLI configured with your credentials
- Docker installed
- These instructions assume you are on x86_64 platform

1. **Set Up Project Structure**

   Create a directory for your Lambda function container image

   ```
   mkdir python-appsignals-container-lambda &&\
   cd python-appsignals-container-lambda
   ```

2. **Create Dockerfile**

   Download and integrate the OpenTelemetry Layer with Application Signals support directly into your Lambda container image. To do this, the `Dockerfile` file is created.

   ```
   # Dockerfile

   FROM public.ecr.aws/lambda/python:3.13

   # Copy function code
   ```

```
COPY app.py ${LAMBDA_TASK_ROOT}

# Install unzip and wget utilities
RUN dnf install -y unzip wget

# Download the OpenTelemetry Layer with AppSignals Support
RUN wget https://github.com/aws-observability/aws-otel-python-instrumentation/
releases/latest/download/layer.zip -O /tmp/layer.zip

# Extract and include Lambda layer contents
RUN mkdir -p /opt && \
    unzip /tmp/layer.zip -d /opt/ && \
    chmod -R 755 /opt/ && \
    rm /tmp/layer.zip

# Set the CMD to your handler
CMD [ "app.lambda_handler" ]
```

> **Note**
>
> The `layer.zip` file contains the OpenTelemetry instrumentation necessary for AWS Application Signals support to monitor your Lambda function.
> The layer extraction steps ensures:
>
> - The layer.zip contents are properly extracted to the `/opt/ directory`
> - The `otel-instrument` script receives proper execution permissions
> - The temporary layer.zip file is removed to keep the image size smaller

3. **Lambda function code**

Create your Lambda function in an `app.py` file:

```
import json
import boto3

def lambda_handler(event, context):
    """
    Sample Lambda function that can be used in a container image.

    Parameters:
    -----------
```

```
    event: dict
        Input event data
    context: LambdaContext
        Lambda runtime information

    Returns:
    --
    dict
        Response object
    """
    print("Received event:", json.dumps(event, indent=2))

    # Create S3 client
    s3 = boto3.client('s3')

    try:
        # List buckets
        response = s3.list_buckets()

        # Extract bucket names
        buckets = [bucket['Name'] for bucket in response['Buckets']]

        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': 'Successfully retrieved buckets',
                'buckets': buckets
            })
        }
    except Exception as e:
        print(f"Error listing buckets: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({
                'message': f'Error listing buckets: {str(e)}'
            })
        }
```

Your project structure should look something like this:

```
.
### Dockerfile
### app.py
```

```
### instructions.md
```

4. **Build and deploy the container image**

   Set up environment variables

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
AWS_REGION=$(aws configure get region)

# For fish shell users:
# set AWS_ACCOUNT_ID (aws sts get-caller-identity --query Account --output text)
# set AWS_REGION (aws configure get region)
```

   Authenticate with public Amazon ECR

```
aws ecr-public get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin public.ecr.aws
```

   Authenticate with private Amazon ECR

```
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

   Create Amazon ECR repository (if needed)

```
aws ecr create-repository \
    --repository-name lambda-appsignals-demo \
    --region $AWS_REGION
```

   Build, tag, and push your image

```
# Build the Docker image
docker build -t lambda-appsignals-demo .

# Tag the image
docker tag lambda-appsignals-demo:latest $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_REGION.amazonaws.com/lambda-appsignals-demo:latest

# Push the image
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/lambda-appsignals-
demo:latest
```

```
5. Create and Configure the Lambda Function
```

5. **Create and configure the Lambda function**

   Create a new function using the Lambda console.

   Select **Container image** as the deployment option.

   Choose **Browse images** to select your Amazon ECR image.

6. **Testing and verifications – Test your Lambda with a simple event. If the layer integration is successful, your Lambda appears under the Application Signals service map.**

   You will see traces and metrics for your Lambda function in the CloudWatch console.

Troubleshooting

If Application Signals is not working, check the following:

- Check the function logs for any errors related to the OpenTelemetry instrumentation
- Verify if the environment variable `AWS_LAMBDA_EXEC_WRAPPER` is set correctly
- Make sure the layer extraction in the Docker file completed successfully
- Confirm if the IAM permissions are properly attached
- If needed, increase the *Timeout and Memory* settings in the general configuration of the Lambda function

Monitor service events

Service Events provides automated deep observability for services monitored with CloudWatch Application Signals. It captures error metrics, function-level performance data, incident snapshots (when requests exceed latency thresholds or throw exceptions), and deployment events — without additional code changes.

How service events works

Service Events collects the following types of signals from your instrumented services:

- **Error metrics** — Per-exception-type error counts and rates for each operation, enabling you to identify which exceptions are most frequent and trending.
- **Function-call metrics** — Invocation count, duration, and error rates for individual functions within your application code.
- **Incident snapshots** — Detailed captures triggered when a request exceeds a latency threshold or throws an exception, including stack traces, call trees, caller details, and operation context.
- **Deployment events** — Markers emitted at application startup and every 24 hours that correlate code deployments with changes in service behavior. The application emits deployment events automatically. Providing deployment metadata (git commit, deployment ID) enriches these events with additional context.

Service Events is automatically enabled when you enable CloudWatch Application Signals for your service. Error metrics and exception tracking are active immediately. Function-call metrics require additional configuration — you must configure packages to instrument before function-call data is collected (see the section called “Enable function instrumentation”). Service Events can be disabled by setting `OTEL_AWS_SERVICE_EVENTS_ENABLED=false`. Data flows from the ADOT SDK to the CloudWatch agent. The agent publishes events to CloudWatch Logs (`/aws/service-events/`*`service-name`* log groups) and CloudWatch Metrics.

Supported languages: Java, Python, and Node.js.

> **Note**
>
> Service Events is automatically disabled in Lambda environments.

Data storage

Service Events stores data in CloudWatch Logs. CloudWatch publishes service event data to a log group with the prefix `/aws/service-events/`*`service-name`*, where *`service-name`* is the value of your `OTEL_SERVICE_NAME` environment variable. One log group is created per service.

You are billed for log ingestion and storage at standard CloudWatch Logs rates.

View errors in the console

In the CloudWatch console, navigate to **Application Signals**, choose your service, and then choose the **Errors** tab. This tab shows exception metrics for your service.

The tab displays:

- An exception count chart showing error trends over time. Use this to detect which exception types have changed in frequency recently.
- A table listing each exception type, the operation where it occurred, the occurrence count, and the change compared to the previous period.

Select an exception to drill down into details including the stack trace, exception message, and a link to the associated trace.

Errors are grouped by operation, exception type, and top stack frames. Only the most recent representative of each group is shown.

> **Note**
>
> To view error data, at least one `/aws/service-events/`*`service-name`* log group must exist in your account. If no log groups exist, the Errors tab displays an onboarding prompt.

View service events in logs

Service Events data is stored in CloudWatch Logs under log groups with the prefix `/aws/service-events/`*`service-name`*. You can query this data directly using CloudWatch Logs Insights to build custom views, create dashboards, or investigate specific incidents.

To query service events:

1. Open the CloudWatch console and navigate to **Logs Insights**.
2. Select the log group `/aws/service-events/`*`service-name`* for your service.
3. Enter a query to filter and analyze the service events data.

Service events in the CloudWatch Application Signals MCP (Model Context Protocol) server

Service events data is accessible through the CloudWatch Application Signals MCP (Model Context Protocol) server, enabling AI coding assistants and agents to query your service's runtime behavior directly.

Troubleshooting

- Automatically correlate errors in your code with production incident snapshots, including full stack traces and affected endpoints.
- Use incident context (exception types, call paths, trace IDs) to suggest targeted fixes without requiring you to manually navigate dashboards.
- Retrieve deployment events to determine whether a recent release introduced a regression.

Performance improvement

- Query function-level performance data to identify bottlenecks when investigating latency issues.
- Compare function call durations across deployments to pinpoint performance regressions.

For setup and usage instructions, see the Application Signals MCP server on the GitHub website.

Configure service events

Prerequisites

To use service events, make sure you have the minimum required versions of the following components:

1. **Update the ADOT SDK** — Update the AWS Distro for OpenTelemetry (ADOT) instrumentation SDK to the latest version for your language (Java, Python, or Node.js).
2. **Update the Amazon EKS add-on (if applicable)** — If you use the CloudWatch Observability Amazon EKS add-on to instrument your applications, update to the latest version of the add-on.
3. **Update the CloudWatch Agent** — Update to version `1.300069.0` or later of the CloudWatch agent.

If you use Amazon EKS, see Enable your applications on Amazon EKS clusters for add-on setup instructions.

Features enabled by default

If you use CloudWatch Application Signals, the following service events signals are **enabled by default** with no additional configuration required:

- Incident snapshots (triggered on exceptions and latency threshold breaches)

- Error metrics (per-exception-type error counts per operation)
- Deployment events (always emitted; enriched when you provide deployment metadata)
- Function instrumentation (enabled by default, but produces no metrics until you configure packages to instrument)

The following features are **opt-in** and require setting environment variables to produce data:

- Function-level metrics (requires configuring `OTEL_AWS_SERVICE_EVENTS_PACKAGES_INCLUDE`)
- Custom endpoint filtering
- Per-endpoint latency thresholds

General settings

Environment variable	Default	Description
`OTEL_AWS_SERVICE_EVENTS_ENABLED`	Follows CloudWatch Application Signals	Toggle for Service Events. Service Events is automatically enabled when CloudWatch Application Signals is enabled. Set to `false` to explicitly disable.
`OTEL_AWS_SERVICE_EVENTS_SAMPLING_MODE`	`always`	Controls function-call data sampling strategy. Values: `always` (record all function calls), `auto` (let the SDK decide based on load), `never` (disable function-call recording). Only applies when function instrumentation packages are configured.

Enable function instrumentation

Function instrumentation is enabled by default, but it produces no metrics until you configure which packages to instrument. Provide a package allow list to start collecting per-function telemetry:

Environment variable	Default	Description
OTEL_AWS_SERVICE_EVENTS_FUNCTION_INSTRUMENT_ENABLED	true	Enables or disables function-level instrumentation. Set to false to disable entirely.
OTEL_AWS_SERVICE_EVENTS_PACKAGES_INCLUDE	None (required for metrics)	Comma-separated list of package prefixes to instrument. No wildcard needed. For example: Java uses com.myapp , Python uses myapp, Node.js uses src/myapp .
OTEL_AWS_SERVICE_EVENTS_PACKAGES_EXCLUDE	None	Comma-separated list of sub-packages to exclude from instrumentation. Exclude always takes precedence over include. For example, include com.myapp and exclude com.myapp.models to instrument your application code but skip data model classes.

Endpoint filtering

Endpoint filtering controls which endpoints generate endpoint error metrics and incident snapshots. These settings do not affect function instrumentation.

Environment variable	Default	Description
OTEL_AWS_SERVICE_EVENTS_ENDPOINT_INCLUDE_PATTERNS	All endpoints	Comma-separated glob patterns of endpoints to include. Matched against METHOD /route.
OTEL_AWS_SERVICE_EVENTS_ENDPOINT_EXCLUDE_PATTERNS	None	Comma-separated glob patterns of endpoints to exclude. Exclude takes precedence when an endpoint matches both.

Latency thresholds

Use the following environment variables to configure latency thresholds for incident snapshot triggers.

Environment variable	Default	Description
`OTEL_AWS_SERVICE_EVENTS_INCIDENT_SNAPSHOT_DURATION_THRESHOLD_MS`	`5000`	Global latency threshold in milliseconds. Requests exceeding this duration trigger an incident snapshot.
`OTEL_AWS_SERVICE_EVENTS_LATENCY_THRESHOLDS`	None	Per-endpoint latency thresholds that override the global default. Format: `METHOD /route:ms` (for example, `GET /health:200,POST /checkout:8000`).

Rate limiting

Use the following environment variables to control the rate at which service events data is collected and reported.

Environment variable	Default	Description
`OTEL_AWS_SERVICE_EVENTS_INCIDENT_SNAPSHOT_MAX_PER_MINUTE`	`100`	Maximum number of incident snapshots captured per minute.
`OTEL_AWS_SERVICE_EVENTS_INCIDENT_SNAPSHOT_MAX_SAME_ERROR`	`1`	Maximum number of snapshots for the same error per capture window.

Configure deployment events

Deployment events are always emitted at application startup and every 24 hours. Providing deployment metadata enriches these events so you can correlate incidents and performance changes with specific code deployments.

Set the following environment variables on your application containers or processes to provide deployment metadata:

Environment variable	Description
`OTEL_AWS_SERVICE_EVENTS_GIT_COMMIT_SHA`	Git commit SHA of the deployed code.
`OTEL_AWS_SERVICE_EVENTS_GIT_REPO_URL`	URL of the Git repository.
`OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_ID`	Unique identifier for the deployment (for example, a CI/CD pipeline run ID).
`OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_TIMESTAMP`	ISO 8601 timestamp of the deployment.
`OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_URL`	URL of the deployment build or pipeline run.

Configure deployment events with GitHub Actions

In your GitHub Actions workflow, use the built-in environment variables to populate deployment metadata. Add the following to your deployment step or container environment:

```
env:
  OTEL_AWS_SERVICE_EVENTS_GIT_COMMIT_SHA: ${{ github.sha }}
  OTEL_AWS_SERVICE_EVENTS_GIT_REPO_URL: ${{ github.server_url }}/
${{ github.repository }}
  OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_ID: ${{ github.run_id }}
  OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_TIMESTAMP: $(date -u +%Y-%m-%dT%H:%M:%SZ)
```

```
  OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_URL: ${{ github.server_url }}/
${{ github.repository }}/actions/runs/${{ github.run_id }}
```

If you deploy container images, pass these values as environment variables in your task definition or pod spec. You can bake them into the image at build time or inject them at deploy time through your deployment configuration.

Configure deployment events with GitLab CI/CD

In your GitLab CI/CD pipeline, use the predefined CI/CD variables to populate deployment metadata. Add the following to your deployment job:

```
deploy:
  variables:
    OTEL_AWS_SERVICE_EVENTS_GIT_COMMIT_SHA: $CI_COMMIT_SHA
    OTEL_AWS_SERVICE_EVENTS_GIT_REPO_URL: $CI_PROJECT_URL
    OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_ID: $CI_PIPELINE_ID
    OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_TIMESTAMP: $(date -u +%Y-%m-%dT%H:%M:%SZ)
    OTEL_AWS_SERVICE_EVENTS_DEPLOYMENT_URL: $CI_PIPELINE_URL
```

Pass these variables to your application containers at deploy time through your container orchestration platform (for example, as environment variables in your Amazon ECS task definition or Kubernetes deployment manifest).

Debug applications with Dynamic Instrumentation

With Dynamic Instrumentation, you can capture runtime state from a live application without restarting or redeploying. Runtime state includes variable values, method arguments, return values, and stack traces. You define instrumentation configurations that specify where in your code to capture data, and the running agent instruments the application at runtime.

Concepts

Breakpoint

Temporary instrumentation that auto-expires. The default expiration is 24 hours, configurable from 5 minutes to 24 hours. Use breakpoints for debugging and investigation.

Probe

Permanent instrumentation that persists until explicitly deleted. Use probes for ongoing observability.

Snapshot

A point-in-time capture of program state including local variables, arguments, return value, exceptions, and stack trace. Dynamic Instrumentation emits snapshots as log records to CloudWatch Logs.

Location

The code location where instrumentation is applied. Required fields differ by language.

Supported languages

- Java
- Python
- JavaScript or TypeScript

Prerequisites

To use Dynamic Instrumentation, update your instrumentation components to the latest version based on your deployment type:

- **Amazon EKS customers** — Update the Amazon CloudWatch Observability EKS add-on to the latest version. The add-on includes the ADOT SDK and CloudWatch Agent. For more information, see Install the CloudWatch Observability EKS add-on.
- **All other customers** — Update both of the following components:
  - The AWS Distro for OpenTelemetry (ADOT) instrumentation SDK for your language (Java, Python, or Node.js).
  - The CloudWatch Agent to the latest version.

The following conditions must also be met:

- CloudWatch Application Signals must be enabled for your application.
- Set the environment variable `OTEL_AWS_DYNAMIC_INSTRUMENTATION_ENABLED=true` on your application.
- Set the environment variable `OTEL_SERVICE_NAME` to your service name.
- Set the environment variable `OTEL_RESOURCE_ATTRIBUTES=deployment.environment.name=`*`my_deployment_env_name`*.

For existing Application Signals users, the value must match the Environment name for your service as it appears in the Application Signals console.

- The CloudWatch Agent must be running with the Application Signals configuration.
- Dynamic Instrumentation is not supported in Lambda environments.

Add dynamic instrumentation to your application

After you instrument your application (see the section called “Prerequisites”), you create an instrumentation *configuration* that specifies which part of your code you want to introduce dynamic telemetry to. Each configuration defines two things:

1. **Where in the code to monitor** — The code location where the breakpoint or probe is applied.
2. **What data to capture** — The runtime state captured when the breakpoint or probe is executed.

> **Note**
>
> By default, Dynamic Instrumentation captures only limited data. To maximize the value of this feature, consider expanding the capture configuration using the options described in the section called “Capture limits”.

You can create configurations using the AWS CLI or SDK, or by using the Model Context Protocol (MCP) server with an AI coding assistant in your IDE.

Create configurations using the CLI or SDK

Use the AWS CLI or AWS SDK to create instrumentation configurations programmatically.

Specify the code location

The location defines where in your code the instrumentation is applied. The required fields differ by language:

Language	Required Fields	Optional Fields
Java	`CodeUnit` (package), `ClassName` , `MethodName` , `FilePath`	`LineNumber`

Language	Required Fields	Optional Fields
Python	`CodeUnit` (module), `MethodName` , `FilePath`	`LineNumber` , `ClassName`
JavaScript or TypeScript	`FilePath`, `LineNumber`	None. Only line-level breakpoints are supported. Probes and function-level breakpoints are not supported . TypeScript is supported when you provide source maps.

Configure what data to capture

The capture configuration controls what runtime state is collected when the instrumentation fires. Available options:

- `CaptureArguments` — List of method argument names to capture.
- `CaptureReturn` — Capture the return value (boolean).
- `CaptureStackTrace` — Capture the stack trace (boolean).
- `CaptureLocals` — List of local variable names to capture.
- `CaptureLimits` — Control capture depth and size (see the section called “Capture limits”).

Configuration parameters

Key parameters when creating a configuration:

- `instrumentation-type` — `BREAKPOINT` or `PROBE`
- `service` — The service name as reported by Application Signals
- `environment` — The environment name
- `signal-type` — `SNAPSHOT`
- `location` — Code location fields (see above)
- `capture-configuration` — Capture options (see above)

Example

The following example creates a breakpoint on a Java method:

```
aws application-signals create-instrumentation-configuration \
    --instrumentation-type BREAKPOINT \
    --service "my-service" \
    --environment "production" \
    --signal-type SNAPSHOT \
    --location '{
        "CodeLocation": {
            "Language": "Java",
            "CodeUnit": "com.example.service",
            "ClassName": "OrderController",
            "MethodName": "processOrder",
            "FilePath": "OrderController.java"
        }
    }' \
    --capture-configuration '{
        "CodeCapture": {
            "CaptureArguments": ["orderId", "user"],
            "CaptureReturn": true,
            "CaptureStackTrace": true,
            "CaptureLimits": {
                "MaxHits": 100,
                "MaxStringLength": 255,
                "MaxCollectionWidth": 20,
                "MaxObjectDepth": 3,
                "MaxFieldsPerObject": 20,
                "MaxStackFrames": 20
            }
        }
    }'
```

Create configurations using the MCP server

The recommended approach to using Dynamic Instrumentation is through the CloudWatch Application Signals MCP (Model Context Protocol) server. The MCP enables AI coding assistants and agents in your IDE to create, manage, and query Dynamic Instrumentation configurations directly from your development environment.

Using the MCP, your AI assistant can:

- Create breakpoints and probes at specific code locations without leaving your editor.
- Query captured snapshots to inspect runtime variable values and call paths.
- Automatically correlate snapshot data with the code you are working on to suggest fixes.
- Manage the lifecycle of instrumentation configurations (view status, delete expired breakpoints).

For setup and usage instructions, see the Application Signals MCP server on the GitHub website.

Data storage

When a breakpoint or probe fires, Dynamic Instrumentation creates a log group in CloudWatch Logs with the prefix `/aws/application-signals/`*`service-name`* (where *`service-name`* is the value of your `OTEL_SERVICE_NAME` environment variable) and writes captured snapshots as log records to that log group.

If the log group does not already exist, Dynamic Instrumentation creates it automatically the first time a snapshot is emitted. You are billed for log ingestion and storage at standard CloudWatch Logs rates.

View and manage configurations

In the CloudWatch console, navigate to the service detail page and choose the **Instrumentation** tab.

- Toggle between **Breakpoints** and **Probes** to view configurations by type.
- View configuration details including description, capture configuration, location, ARN, and expiration time.
- View status history to track transitions: Ready to Active to Error/Disabled.
- Delete configurations that are no longer needed.

Understand status

Each instrumentation configuration has a status that indicates its current state.

Status	Description
READY	The agent received the configuration.

Status	Description
ACTIVE	The agent applied the instrumentation to the running application.
ERROR	The instrumentation failed to apply. See the error cause for details.
DISABLED	The instrumentation expired, or you removed it.

When an instrumentation enters the ERROR state, the following causes might be reported:

Error Cause	Description
FILE_NOT_FOUND	The specified file path does not exist in the application.
METHOD_NOT_FOUND	The specified method does not exist in the target class or module.
LINE_NOT_EXECUTABLE	The specified line number does not correspond to an executable statement.
OVERLOADED_METHODS	Multiple methods match the specified name. Provide additional location details to identify the correct method.
LANGUAGE_MISMATCH	The location fields do not match the language of the running application.
RUNTIME_ERROR	An unexpected error occurred while applying the instrumentation.

Capture limits

Capture limits control the size and depth of captured data. Configure these values in the `capture-limits` field of the capture configuration.

Limit	Default	Range	Description
maxStringLength	255	1–255	Maximum characters captured per string value.
maxCollectionWidth	20	1–20	Maximum elements captured per collection or array.
maxObjectDepth	3	1–5	Maximum depth for nested object traversal.
maxFieldsPerObject	20	1–20	Maximum fields captured per object.
maxStackFrames	20	1–20	Maximum stack frames captured.
maxHits	100	1–1000	Maximum captures before auto-disabling. Breakpoints only.

Each instrumentation point is rate-limited to 5 captures per second.

Troubleshooting your Application Signals installation

This section contains troubleshooting tips for CloudWatch Application Signals.

Topics

- Address OpenTelemetry configuration conflicts in Amazon EKS with Application Signals
- Application Signals Java layer cold start performance
- Application doesn't start after Application Signals is enabled
- Python application doesn't start after Application Signals is enabled
- No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI)
- My Node.js application is not instrumented or isn't generating Application Signals telemetry
- My .NET application isn't instrumented or breaks for AWS SDK calls
- No application data in Application Signals dashboard

- Service metrics or dependency metrics have Unknown values
- Handling a ConfigurationConflict when managing the Amazon CloudWatch Observability EKS add-on
- I want to filter out unnecessary metrics and traces
- What does InternalOperation mean?
- How do I enable logging for .NET applications?
- How can I resolve assembly version conflicts in .NET applications?
- Can I disable FluentBit?
- Can I filter container logs before exporting to CloudWatch Logs?
- Resolving TypeError when Using AWS Distro for OpenTelemetry (ADOT) JavaScript Lambda Layer
- TypeError when using Response Streaming Lambda handlers with AWS Distro for OpenTelemetry (ADOT) JavaScript Lambda Layer
- Update to required versions of agents or Amazon EKS add-on
- Embedded Metric Format (EMF) disabled for Application Signals

Address OpenTelemetry configuration conflicts in Amazon EKS with Application Signals

If you use OpenTelemetry (OTel) for application performance monitoring (APM) with Amazon EKS and configure custom OTLP exporter endpoints other than CloudWatch endpoints, you may experience the following behaviors after installing or upgrading to CloudWatch Observability add-on version 5.0.0 or later:

- Disruption to existing OTel telemetry – CloudWatch Observability add-on may override OTLP exporter endpoints that you hard coded in your application. This override doesn't affect endpoints configured through container environment variables or `envFrom` ConfigMap. When overridden, your metrics and traces may not reach their intended destination. To maintain your existing APM setup after upgrading to V5.0.0 or later, see Opt out of Application Signals
- Application Signals may not work if you previously enabled Application Signals using CloudWatch Observability add-on and have a custom OTLP endpoint configured. To resolve this, either remove custom OTLP endpoints or set the environment variable `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=true` when installing or upgrading to version 5.0.0 or later

Application Signals Java layer cold start performance

Adding the Application Signals Layer to Java Lambda functions increases the startup latency (cold start time). The following tips can help reduce latency for time-sensitive functions.

Fast startup for Java agent – The Application Signals Java Lambda Layer includes a Fast Startup feature that's turned off by default but can be enabled by setting the OTEL_JAVA_AGENT_FAST_STARTUP_ENABLED variable to true. When enabled, this feature configures the JVM to use tiered compilation level 1 C1 compiler to generate quick optimized native code for faster cold starts. The C1 compiler prioritizes speed at the cost of long-term optimization whereas the C2 compiler provides superior overall performance by profiling data over time.

For more information, see Fast startup for Java agent .

Reduce cold start times with Provisioned Concurrency – AWS Lambda provisioned concurrency pre-allocates a specified number of function instances, keeping them initialized and ready to handle requests immediately. This reduces cold-start times by eliminating the need to initialize the function environment during execution, ensuring faster and more consistent performance, especially for latency-sensitive workloads. For more information, see Configuring provisioned concurrency for a function .

Optimize startup performance using Lambda SnapStart – AWS Lambda SnapStart is a feature that optimizes the startup performance of Lambda functions by creating a pre-initialized snapshot of the execution environment after the function's initialization phase. This snapshot is then reused to start new instances, significantly reducing cold-start times by skipping the initialization process during function invocation. For information, see Improving startup performance with Lambda SnapStart

Application doesn't start after Application Signals is enabled

If your application on an Amazon EKS cluster doesn't start after you enable Application Signals on the cluster, check for the following:

- Check if the application has been instrumented by another monitoring solution. Application Signals might not support co-existing with other instrumentation solutions.
- Confirm that your the application meets the compatibility requirements to use Application Signals. For more information, see Supported systems.

- If your application failed to pull the Application Signals artifacts such as the AWS Distro for OpenTelemetery Java or Python agent and CloudWatch agent images, it could be a network issue.

To mitigate the issue, remove the annotation `instrumentation.opentelemetry.io/inject-java: "true"` or `instrumentation.opentelemetry.io/inject-python: "true"` from your application deployment manifest, and re-deploy your application. Then check if the application is working.

Known issues

The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.5.0-eksbuild.1`.

If you are impacted, either downgrade the version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Python application doesn't start after Application Signals is enabled

It is a known issue in OpenTelemetry auto-instrumentation that a missing `PYTHONPATH` environment variable can sometimes cause the application to fail to start . To resolve this, make sure that you set the `PYTHONPATH` environment variable to the location of your application's working directory. For more information about this issue, see Python autoinstrumentation setting of PYTHONPATH is not compliant with Python's module resolution behavior, breaking Django applications.

For Django applications, there are additional required configurations, which are outlined in the OpenTelemetry Python documentation.

- Use the `--noreload` flag to prevent automatic reloading.
- Set the `DJANGO_SETTINGS_MODULE` environment variable to the location of your Django application's `settings.py` file. This ensures that OpenTelemetry can correctly access and integrate with your Django settings.

No Application Signals data for a Python application that uses a pre-fork server (WSGI or ASGI)

Pre-fork servers, such as Gunicorn, uWSGI, or Uvicorn with multiple workers, fork worker processes from a main process. The OpenTelemetry SDK relies on background threads that don't survive a fork, so when the main process is instrumented, the workers can't export traces, metrics, and logs.

For more background about this OpenTelemetry behavior, see Pre-fork server issues in the OpenTelemetry Python documentation.

To resolve this, you must first initialize the auto-instrumentation in each forked worker process rather than in the main process, and then configure ADOT Python to skip the main process.

> **Note**
>
> Be sure that you are using the latest version of ADOT Python and the Amazon CloudWatch Observability EKS add-on before proceeding.

Additional steps to enable Application Signals with a pre-fork server

1. Initialize the auto-instrumentation in the forked worker processes.

   For Gunicorn, use the `post_fork` hook:

   ```
   # gunicorn.conf.py
   def post_fork(server, worker):
       from opentelemetry.instrumentation.auto_instrumentation import sitecustomize
   ```

   For uWSGI, use the `import` directive.

   ```
   #  uwsgi.ini
   [uwsgi]
   ; required for the instrumentation of worker processes
   enable-threads = true
   lazy-apps = true
   import = opentelemetry.instrumentation.auto_instrumentation.sitecustomize
   ```

For Uvicorn, FastAPI, or other ASGI applications that you run with multiple workers, initialize the instrumentation inside the worker by calling `initialize()` before you import your application. Place the following at the top of your application's entry-point module:

```
from opentelemetry.instrumentation.auto_instrumentation import initialize

initialize()

# Import your application AFTER initialize() so that it is instrumented.
from fastapi import FastAPI

app = FastAPI()
```

Then start the server as usual, for example:

```
uvicorn main:app --workers 2
```

Alternatively, if you run an ASGI application under Gunicorn, you can use Gunicorn's `UvicornWorker`, which preserves background threads across the fork, together with the `post_fork` hook shown previously:

```
opentelemetry-instrument gunicorn \
  --workers 4 \
  --worker-class uvicorn.workers.UvicornWorker \
  --bind 0.0.0.0:8000 \
  myapp.main:app
```

2. Enable the configuration for ADOT Python auto-instrumentation to skip the main process and defer to workers by setting the `OTEL_AWS_PYTHON_DEFER_TO_WORKERS_ENABLED` environment variable to `true`.

   With both steps applied, each worker process initializes its own tracer, metric, and log pipelines after the fork, so that traces, metrics, and logs are exported correctly.

My Node.js application is not instrumented or isn't generating Application Signals telemetry

To enable Application Signals for Node.js, you must make sure that your Node.js application uses the CommonJS (CJS) module format. The AWS Distro for OpenTelemetry Node.js doesn't support the ESM module format, because OpenTelemetry JavaScript's support of ESM is experimental and is a work in progress.

To determine if your application is using CJS and not ESM, make sure that your application does not fulfill the conditions to enable ESM.

My .NET application isn't instrumented or breaks for AWS SDK calls

The AWS Distro for Open Telemetry (ADOT) SDK for .NET does not support AWS SDK for .NET V4. Use AWS SDK .NET V3 for full Application Signals support.

No application data in Application Signals dashboard

If metrics or traces are missing in the Application Signals dashboards, the following might be causes. Investigate these causes only if you have waited 15 minutes for Application Signals to collect and display data since your last update.

- Make sure that your library and framework you are using is supported by the ADOT Java agent. For more information, see Libraries / Frameworks.
- Make sure that the CloudWatch agent is running. First check the status of the CloudWatch agent pods and make sure they are all in `Running` status.

  ```
  kubectl -n amazon-cloudwatch get pods.
  ```

  Add the following to the CloudWatch agent configuration file to enable debugging logs, and then restart the agent.

  ```
  "agent": {

    "region": "${REGION}",
    "debug": true
  },
  ```

  Then check for errors in the CloudWatch agent pods.

- Check for configuration issues with the CloudWatch agent. Confirm that the following is still in the CloudWatch agent configuration file and the agent has been restarted since it was added.

  ```
  "agent": {
    "region": "${REGION}",
    "debug": true
  },
  ```

  Then check the OpenTelemetry debugging logs for error messages such as `ERROR io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export ...`. These messages might indicate the problem.

  If that doesn't solve the issue, dump and check the environment variables with names that start with `OTEL_` by describing the pod with the `kubectl describe pod` command.

- To enable the OpenTelemetry Python debug logging, set the environment variable `OTEL_PYTHON_LOG_LEVEL` to `debug` and redeploy the application.
- Check for wrong or insufficient permissions for exporting data from the CloudWatch agent. If you see `Access Denied` messages in the CloudWatch agent logs, this might be the issue. It is possible that the permissions applied when you installed the CloudWatch agent were later changed or revoked.
- Check for an AWS Distro for OpenTelemetry (ADOT) issue when generating telemetry data.

  Make sure that the instrumentation annotations `instrumentation.opentelemetry.io/inject-java` and `sidecar.opentelemetry.io/inject-java` are applied to the application deployment and the value is `true`. Without these, the application pods will not be instrumented even if the ADOT addon is installed correctly.

  Next, check if the `init` container is applied on the application and the `Ready` state is `True`. If the `init` container is not ready, see the status for the reason.

  If the issue persists, enable debug logging on the OpenTelemetry Java SDK by setting the environment variable `OTEL_JAVAAGENT_DEBUG` to true and redeploying the application. Then look for messages that start with `ERROR io.telemetry`.

- The metric/span exporter might be dropping data. To find out, check the application log for messages that include `Failed to export...`
- The CloudWatch agent might be getting throttled when sending metrics or spans to Application Signals. Check for messages indicating throttling in the CloudWatch agent logs.

- Make sure that you've enabled the service discovery setup. You need to do this only once in your Region.

  To confirm this, in the CloudWatch console choose **Application Signals**, **Services**. If Step 1 is not marked **Complete**, choose **Start discovering your services**. Data should start flowing in within five minutes.

Service metrics or dependency metrics have Unknown values

If you see **UnknownService**, **UnknownOperation**, **UnknownRemoteService**, or **UnknownRemoteOperation** for a dependency name or operation in the Application Signals dashboards, check whether the occurrence of data points for the unknown remote service and unknown remote operation are coinciding with their deployments.

- **UnknownService** means that the name of an instrumented application is unknown. If the `OTEL_SERVICE_NAME` environment variable is undefined and `service.name` isn't specified in `OTEL_RESOURCE_ATTRIBUTES`, the service name is set to `UnknownService`. To fix this, specify the service name in `OTEL_SERVICE_NAME` or `OTEL_RESOURCE_ATTRIBUTES`.
- **UnknownOperation** means that the name of an invoked operation is unknown. This occurs when Application Signals is unable to discover an operation name which invokes the remote call, or when the extracted operation name contains high cardinality values.
- **UnknownRemoteService** means that the name of the destination service is unknown. This occurs when the system is unable to extract the destination service name that the remote call accesses.

  One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.service` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteService`. For more information about customizations in the CloudWatch agent, see Enable CloudWatch Application Signals.
- **UnknownRemoteOperation** means that the name of the destination operation is unknown. This occurs when the system is unable to extract the destination operation name that the remote call accesses.

  One solution is to create a custom span around the function that sends out the request, and add the attribute `aws.remote.operation` with the designated value. Another option is to configure the CloudWatch agent to customize the metric value of `RemoteOperation`. For more

information about customizations in the CloudWatch agent, see Enable CloudWatch Application Signals.

Handling a ConfigurationConflict when managing the Amazon CloudWatch Observability EKS add-on

When you install or update the Amazon CloudWatch Observability EKS add-on, if you notice a failure caused by a `Health Issue` of type `ConfigurationConflict` with a description that starts with `Conflicts found when trying to apply. Will not continue due to resolve conflicts mode`, it is likely because you already have the CloudWatch agent and its associated components such as the ServiceAccount, the ClusterRole and the ClusterRoleBinding installed on the cluster. When the add-on tries to install the CloudWatch agent and its associated components, if it detects any change in the contents, it by default fails the installation or update to avoid overwriting the state of the resources on the cluster.

If you are trying to onboard to the Amazon CloudWatch Observability EKS add-on and you see this failure, we recommend deleting an existing CloudWatch agent setup that you had previously installed on the cluster and then installing the EKS add-on. Be sure to back up any customizations you might have made to the original CloudWatch agent setup such as a custom agent configuration, and provide these to the Amazon CloudWatch Observability EKS add-on when you next install or update it. If you had previously installed the CloudWatch agent for onboarding to Container Insights, see Setup guide (AWS CLI) for more information.

Alternatively, the add-on supports a conflict resolution configuration option that has the capability to specify `OVERWRITE`. You can use this option to proceed with installing or updating the add-on by overwriting the conflicts on the cluster. If you are using the Amazon EKS console, you'll find the **Conflict resolution method** when you choose the **Optional configuration settings** when you create or update the add-on. If you are using the AWS CLI, you can supply the `--resolve-conflicts OVERWRITE` to your command to create or update the add-on.

I want to filter out unnecessary metrics and traces

If Application Signals is collecting traces and metrics that you don't want, see Manage high-cardinality operations for information about configuring the CloudWatch agent with custom rules to reduce cardinality.

For information about customizing trace sampling rules, see Configure sampling rules in the X-Ray documentation.

What does `InternalOperation` mean?

An `InternalOperation` is an operation that is triggered by the application internally rather than by an external invocation. Seeing `InternalOperation` is expected, healthy behavior.

Some typical examples where you would see `InternalOperation` include the following:

- **Preloading on start**– Your application performs an operation named `loadDatafromDB` which reads metadata from a database during the warm up phase. Instead of observing `loadDatafromDB` as a service operation, you'll see it categorized as an `InternalOperation`.
- **Async execution in the background**– Your application subscribes to an event queue, and processes streaming data accordingly whenever there's an update. Each triggered operation will be under `InternalOperation` as a service operation.
- **Retrieving host information from a service registry**– Your application talks to a service registry for service discovery. All interactions with the discovery system are classified as an `InternalOperation`.

How do I enable logging for .NET applications?

To enable logging for .NET applications, configure the following environment variables. For more information about how to configure these environment variables, see Troubleshooting .NET automatic instrumentation issues in the OpenTelemetry documentation.

- `OTEL_LOG_LEVEL`
- `OTEL_DOTNET_AUTO_LOG_DIRECTORY`
- `COREHOST_TRACE`
- `COREHOST_TRACEFILE`

How can I resolve assembly version conflicts in .NET applications?

If you get the following error, see Assembly version conflicts in the OpenTelemetry documentation for resolution steps.

```
Unhandled exception. System.IO.FileNotFoundException: Could not load file or
 assembly 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'. The system cannot find the file
 specified.
```

```
File name: 'Microsoft.Extensions.DependencyInjection.Abstractions, Version=7.0.0.0,
 Culture=neutral, PublicKeyToken=adb9793829ddae60'
   at Microsoft.AspNetCore.Builder.WebApplicationBuilder..ctor(WebApplicationOptions
 options, Action`1 configureDefaults)
   at Microsoft.AspNetCore.Builder.WebApplication.CreateBuilder(String[] args)
   at Program.<Main>$(String[] args) in /Blog.Core/Blog.Core.Api/Program.cs:line 26
```

Can I disable FluentBit?

You can disable FluentBit by configuring the Amazon CloudWatch Observability EKS add-on. For more information, see (Optional) Additional configuration.

Can I filter container logs before exporting to CloudWatch Logs?

No, filtering container logs is not yet supported.

Resolving TypeError when Using AWS Distro for OpenTelemetry (ADOT) JavaScript Lambda Layer

Your Lambda function may fail with this error: `TypeError - "Cannot redefine property: handler"` when you:

- Use the ADOT JavaScript Lambda Layer
- Use `esbuild` to compile TypeScript
- Export your handler with the `export` keyword

The ADOT JavaScript Lambda Layer needs to modify your handler at runtime. When you use the `export` keyword with `esbuild` (directly or through AWS CDK), `esbuild` makes your handler immutable, preventing these modifications.

Export your handler function using `module.exports` instead of the `export` keyword:

```
// Before
export const handler = (event) => {
  // Handler Code
}
```

```
// After
```

```
const handler = async (event) => {
  // Handler Code
}
module.exports = { handler }
```

TypeError when using Response Streaming Lambda handlers with AWS Distro for OpenTelemetry (ADOT) JavaScript Lambda Layer

Your Lambda function may fail with this error: `TypeError - "responseStream.write is not a function"` when you:

- Use the ADOT JavaScript Lambda Layer with AWS Lambda Instrumentation enabled (enabled by default)
- Using the response streaming feature in Node.js managed runtimes. For example, when your function handler is like:

```
* export const handler = awslambda.streamifyResponse(...)
```

The AWS Lambda Instrumentation in the ADOT JavaScript Lambda Layer currently does not support Response Streaming in Node.js managed runtimes, so it must be disabled to avoid this TypeError.

Update to required versions of agents or Amazon EKS add-on

After August 9, 2024, CloudWatch Application Signals will no longer support older versions of the Amazon CloudWatch Observability EKS add-on, the CloudWatch agent, and the AWS Distro for OpenTelemetry auto-instrumentation agent.

- For the Amazon CloudWatch Observability EKS add-on, versions older than `v1.7.0-eksbuild.1` won't be supported.
- For the CloudWatch agent, versions older than `1.300040.0` won't be supported.
- For the AWS Distro for OpenTelemetry auto-instrumentation agent:
  - For Java, versions older than `1.32.2` aren't supported.
  - For Python, versions older than `0.2.0` aren't supported.
  - For .NET, versions older than `1.3.2` aren't supported.
  - For Node.js, versions older than `0.3.0` aren't supported.

Important

The latest versions of the agents include updates to the Application Signals metric schema. These updates are not backward compatible, and this can result in data issues if incompatible versions are used. To help ensure a seamless transition to the new functionality, do the following:

- If your application is running on Amazon EKS, be sure to restart all instrumented applications after you update the Amazon CloudWatch Observability add-on.
- For applications running on other platforms, be sure to upgrade **both** the CloudWatch agent and the AWS OpenTelemetry auto-instrumentation agent to the latest versions.

The instructions in the following sections can help you update to a supported version.

Contents

- Update the Amazon CloudWatch Observability EKS add-on
  - Use the console
  - Use the AWS CLI
- Update the CloudWatch agent and ADOT agent
  - Update on Amazon ECS
  - Update on Amazon EC2 or other architectures

Update the Amazon CloudWatch Observability EKS add-on

To the Amazon CloudWatch Observability EKS add-on, you can use the AWS Management Console or the AWS CLI.

Use the console

To upgrade the add-on using the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of the Amazon EKS cluster to update.
3. Choose the **Add-ons** tab, then choose **Amazon CloudWatch Observability**.
4. Choose **Edit**, select the version you want to update to, and then choose **Save changes**.

Be sure to choose `v1.7.0-eksbuild.1` or later.

5. Enter one of the following AWS CLI commands to restart your services.

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

Use the AWS CLI

To upgrade the add-on using the AWS CLI

1. Enter the following command to find the latest version.

```
aws eks describe-addon-versions \
--addon-name amazon-cloudwatch-observability
```

2. Enter the following command to update the add-on. Replace *$VERSION* with a version that is `v1.7.0-eksbuild.1` or later. Replace *$AWS_REGION* and *$CLUSTER* with your Region and cluster name.

```
aws eks update-addon \
--region $AWS_REGION \
--cluster-name $CLUSTER \
--addon-name amazon-cloudwatch-observability \
--addon-version $VERSION \
# required only if the advanced configuration is used.
--configuration-values $JSON_CONFIG
```

> **Note**
>
> If you're using an custom configuration for the add-on, you can find an example of the configuration to use for *$JSON_CONFIG* in Enable CloudWatch Application Signals.

3. Enter one of the following AWS CLI commands to restart your services.

```
# Restart a deployment
kubectl rollout restart deployment/name
# Restart a daemonset
kubectl rollout restart daemonset/name
# Restart a statefulset
kubectl rollout restart statefulset/name
```

Update the CloudWatch agent and ADOT agent

If your services are running on architectures other than Amazon EKS, you will need to upgrade both the CloudWatch agent and the ADOT auto-instrumentation agent to use the latest Application Signals features.

Update on Amazon ECS

To upgrade your agents for services running on Amazon ECS

1. Create a new task definition revision. For more information, see Updating a task definition using the console.
2. Replace the `$IMAGE` of the `ecs-cwagent` container with the latest image tag from cloudwatch-agent on Amazon ECR.

   If you upgrade to a fixed version, be sure to use a version equal to or later than `1.300040.0`.
3. Replace the `$IMAGE` of the `init` container with the latest image tag from the following locations:

   - For Java, use aws-observability/adot-autoinstrumentation-java.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `1.32.2`.
   - For Python, use aws-observability/adot-autoinstrumentation-python.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `0.2.0`.
   - For .NET, use aws-observability/adot-autoinstrumentation-dotnet.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `1.3.2`.
   - For Node.js, use aws-observability/adot-autoinstrumentation-node.

     If you upgrade to a fixed version, be sure to use a version equal to or later than `0.3.0`.

4. Update the Application Signals environment variables in your app container by following the instructions at Step 4: Instrument your application with the CloudWatch agent.
5. Deploy your service with the new task definition.

Update on Amazon EC2 or other architectures

To upgrade your agents for services running on Amazon EC2 or other architectures

1. Be sure to select version `1.300040.0` or later of the CloudWatch agent version.
2. Download the latest version of the AWS Distro for OpenTelemetry auto-instrumentation agent from one of the following locations:
   - For Java, use aws-otel-java-instrumentation .

     If you upgrade to a fixed version, be sure to choose `1.32.2` or later.
   - For Python, use aws-otel-python-instrumentation .

     If you upgrade to a fixed version, be sure to choose `0.2.0` or later.
   - For .NET, use aws-otel-dotnet-instrumentation.

     If you upgrade to a fixed version, be sure to choose `1.3.2` or later.
   - For Node.js, use aws-otel-js-instrumentation .

     If you upgrade to a fixed version, be sure to choose `0.3.0` or later.
3. Apply the updated Application Signals environment variables to your application, then start your application. For more information, see Step 3: Instrument your application and start it.

Embedded Metric Format (EMF) disabled for Application Signals

Disabling EMF for the `/aws/application-signals/data` log group can have the following impact on Application Signals functionality.

- Application Signals metrics and charts will not be displayed
- Application Signals functionality will be degraded

How do I restore Application Signals?

When Application Signals displays empty charts or metrics, you must enable EMF for the `/aws/application-signals/data` log group to restore full functionality. For more information, see PutAccountPolicy.

(Optional) Configuring Application Signals

This section contains information about configuring CloudWatch Application Signals.

Topics

- Trace sampling rate
- Enable trace to log correlation
- Enable metric to log correlation
- Configure HTTP endpoint visibility
- Manage high-cardinality operations

Trace sampling rate

By default, when you enable Application Signals X-Ray centralized sampling is enabled using the default sampling rate settings of `reservoir=1/s` and `fixed_rate=5%`. The environment variables for the AWS Distro for OpenTelemetry (ADOT) SDK agent as set as follows.

Environment variable	Value	Note
`OTEL_TRACES_SAMPLER`	`xray`	
`OTEL_TRACES_SAMPLER_ARG`	`endpoint=http://cloudwatch-agent.amazon-cloudwatch:2000`	Endpoint of the CloudWatch agent

For information about changing the sampling configuration, see the following:

- To change X-Ray sampling, see Configure sampling rules
- To change ADOT sampling, see Configuring the OpenTelemetry Collector for X-Ray remote sampling

If you want to disable X-Ray centralized sampling and use local sampling instead, set the following values for the ADOT SDK Java agent as below. The following example sets the sampling rate at 5%.

Environment variable	Value
`OTEL_TRACES_SAMPLER`	`parentbased_traceidratio`
`OTEL_TRACES_SAMPLER_ARG`	`0.05`

For information about more advanced sampling settings, see OTEL_TRACES_SAMPLER.

Enable trace to log correlation

You can enable *trace to log correlation* in Application Signals. This automatically injects trace IDs and span IDs into the relevant application logs. Then, when you open a trace detail page in the Application Signals console, the relevant log entries (if any) that correlate with the current trace automatically appear at the bottom of the page.

For example, suppose you notice a spike in a latency graph. You can choose the point on the graph to load the diagnostics information for that point in time. You then choose the relevant trace to get more information. When you view the trace information, you can scroll down to see the logs associated with the trace. These logs might reveal patterns or error codes associated with the issues causing the latency spike.

Instrument your application before configuring trace log correlation

Your application must already be instrumented for Application Signals before you configure trace log correlation. The Application Signals instrumentation is what populates the `trace_id`, `span_id`, and `trace_flags` Mapped Diagnostic Context (MDC) values at runtime. Without instrumentation, these values remain empty (for example, `trace_id=`) and correlation does not work.
For instrumentation setup instructions, see the enablement guide for your architecture: Enable your applications on Amazon EKS clusters, Enable your applications on Amazon ECS, or Enable your applications on Amazon EC2.

To achieve trace log correlation, Application Signals relies on the following:

- Logger MDC auto-instrumentation for Java.
- OpenTelemetry Logging Instrumentation for Python.
- The Pino, Winston, or Bunyan auto-instrumentations for Node.js.

The OpenTelemetry community provides all of these instrumentations. Application Signals uses them to inject trace contexts such as trace ID and span ID into application logs. To enable this, you must manually change your logging configuration to enable the auto-instrumentation.

Depending on the architecture that your application runs on, you might have to also set an environment variable to enable trace log correlation, in addition to following the steps in this section.

- On Amazon EKS, no additional environment variable is needed. You still must configure your logging output as described in the following examples, and your logger must write to stdout so that Container Insights can collect the logs.
- On Amazon ECS, no additional environment variable is needed. You still must configure your logging output as described in the following examples, and your logger must write to stdout so that Container Insights can collect the logs.
- On Amazon EC2, see the step 4 in the procedure in Step 3: Instrument your application and start it.

OpenTelemetry Container Insights does not enable trace log correlation automatically

Enabling the OpenTelemetry-based Container Insights log collection path (for example, through the CloudWatch Observability EKS add-on) collects and ships your container logs to CloudWatch Logs, but it does not instrument your application for Application Signals or inject trace context into those logs. To correlate traces with logs, you must separately enable Application Signals instrumentation *and* configure your logging pattern as described on this page.

After you enable trace log correlation, when you open a trace detail page, any log entries that contain the same trace ID automatically appear at the bottom of the page. With trace log correlation, you can quickly move from a trace to the relevant application logs without manually searching.

Trace log correlation setup examples

This section contains examples of setting up trace log correlation in several environments.

Spring Boot for Java

Suppose you have a Spring Boot application in a folder called `custom-app`. The application configuration is usually a YAML file named `custom-app/src/main/resources/application.yml` that might look like this:

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...
```

To enable trace log correlation, add the following logging configuration.

```
spring:
  application:
    name: custom-app
  config:
    import: optional:configserver:${CONFIG_SERVER_URL:http://localhost:8888/}

...

logging:
  pattern:
    level: trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags}
 %5p
```

Spring Boot and XML configuration are alternatives

The Spring Boot `application.yml` (`logging.pattern.level`) approach and the Logback/Log4j2/Log4j XML configuration approaches shown in the following sections are *alternatives*. Use one or the other, not both. If you set the pattern in `application.yml`, Spring Boot's default Logback configuration picks it up automatically and no separate `logback.xml` is needed.

EKS and ECS: Write logs to stdout

Container Insights collects logs from the container's stdout/stderr, not from files written inside the container. Configure your logger to write to the console (stdout). A FileAppender writing to a file such as `app.log` produces correctly trace-tagged lines, but they are not collected and do not appear correlated on the trace detail page. If you must log to a file, you additionally need to ship that file (for example, a sidecar collector reading a shared volume), which is outside the scope of this page.

Logback for Java

In the logging configuration (such as logback.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `pattern` of Encoder. For example, the following configuration prepends the trace context before the log message.

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
  <encoder>
    <pattern>trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n</pattern>
  </encoder>
</appender>
```

For more information about encoders in Logback, see Encoders in the Logback documentation.

Log4j2 for Java

In the logging configuration (such as log4j2.xml), insert the trace context `trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=%mdc{trace_flags} %5p` into `PatternLayout`. For example, the following configuration prepends the trace context before the log message.

```
<Appenders>
  <Console name="CONSOLE" target="SYSTEM_OUT">
    <PatternLayout pattern="trace_id=%mdc{trace_id} span_id=%mdc{span_id} trace_flags=
%mdc{trace_flags} %5p - %m%n"/>
  </Console>
</Appenders>
```

For more information about pattern layouts in Log4j2, see Pattern Layout in the Log4j2 documentation.

Log4j for Java

In the logging configuration (such as log4j.xml), insert the trace context `trace_id=%X{trace_id} span_id=%X{span_id} trace_flags=%X{trace_flags} %5p` into `PatternLayout`. For example, the following configuration prepends the trace context before the log message.

```
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="trace_id=%X{trace_id} span_id=%X{span_id}
 trace_flags=%X{trace_flags} %5p - %m%n"/>
  </layout>
</appender>
```

For more information about pattern layouts in Log4j, see Class Pattern Layout in the Log4j documentation.

Python

Set the environment variable `OTEL_PYTHON_LOG_CORRELATION` to `true` while running your application. For more information, see Enable trace context injectionin the Python OpenTelemetry documentation.

Node.js

For more information about enabling trace context injection in Node.js for the logging libraries that support it, see the NPM usage documentations of the Pino, Winston, or Bunyan auto-instrumentations for Node.js.

Verify the setup

After redeploying your application, confirm that a log line now shows a populated `trace_id` (for example, by running `kubectl logs` *`pod-name`*). Then open a trace detail page in the Application Signals console and confirm that the correlated log entries appear at the bottom of the page.

On Amazon EKS, logs are written to the Container Insights application log group (`/aws/containerinsights/`*`cluster-name`*`/application`, or `/aws/otel/containerinsights/`*`cluster-name`*`/application` when the OpenTelemetry Container Insights log path is enabled).

Enable metric to log correlation

If you publish application logs to log groups in CloudWatch Logs, you can enable *metric to application log correlation* in Application Signals. With metric log correlation, the Application Signals console automatically displays the relevant log groups associated with a metric.

For example, suppose you notice a spike in a latency graph. You can choose a point on the graph to load the diagnostics information for that point in time. The diagnostics information will show the relevant application log groups that are associated with the current service and metric. Then you can choose a button to run a CloudWatch Logs Insights query on those log groups. Depending on the information contained in the application logs, this might help you to investigate the cause of the latency spike.

Depending on the architecture that your application runs on, you might have to also set an environment variable to enable metric to application log correlation.

- On Amazon EKS, no further steps are needed.
- On Amazon ECS, no further steps are needed.
- On Amazon EC2, see step 4 in the procedure in Step 3: Instrument your application and start it.

Configure HTTP endpoint visibility

This page provides guidance for Application Signals customers who expect to see their HTTP service operations separated by specific endpoint names, but instead see multiple endpoints grouped into the same metric. By default, Application Signals truncates the `Operation` metric dimension to the first URL path segment for HTTP services (for example, `/api/v1/users` becomes `GET /api`) to preserve low-cardinality metrics. As a result, users may find that services with multiple endpoints sharing the same prefix have limited visibility into the operational health of individual endpoints. You can follow the steps below to configure service operation endpoints to your desired granularity.

> **Important**
>
> Modifying endpoint visibility is a breaking change as it affects the Application Signals metric dimensions for service operations. Remember to update your SLO thresholds, alarms, and/or dashboards accordingly.

AWS Distro for OpenTelemetry (ADOT) solution

For customers using AWS Distro for OpenTelemetry (ADOT), set the `OTEL_AWS_HTTP_OPERATION_PATHS` environment variable with a comma-separated list of URL path templates for the HTTP service:

```
export OTEL_AWS_HTTP_OPERATION_PATHS="/path/to/endpoint, /another/{placeholder}/
endpoint"
```

This variable uses the longest matching prefix against the HTTP server span's `url.path` attribute to determine the operation name. Wildcard patterns match any single URL segment and can be denoted by `{placeholder}`, `:placeholder`, or `*`. After setting the variable, restart your application for the new endpoint groupings to take effect.

This variable is supported in ADOT for Java, Python, Node.js, and .NET.

Example

Consider an API service that receives the following traffic:

```
GET  /api/users
GET  /api/users/42
GET  /api/users/42/orders
POST /api/users/99/orders
POST /api/users/42/orders
GET  /api/users/42/orders/7/items
GET  /api/products
```

By default, Application Signals groups all of these into `GET /api` or `POST /api` service metrics, making it impossible to distinguish performance between endpoints.

To fix this, set the environment variable with the desired path templates:

```
export OTEL_AWS_HTTP_OPERATION_PATHS="/api/users/{userId}/orders/{orderId}/items, /api/
users/{userId}/orders, /api/users/{userId}, /api/users, /api/products"
```

With this configuration, Application Signals shows distinct operations. Multiple requests can resolve to the same configured template:

Incoming request	Default operation	With config
`GET /api/users`	`GET /api`	`GET /api/users`
`GET /api/users/42`	`GET /api`	`GET /api/users/{userId}`
`GET /api/users/42/orders`	`GET /api`	`GET /api/users/{userId}/orders`
`POST /api/users/99/orders`	`POST /api`	`POST /api/users/{userId}/orders`
`POST /api/users/42/orders`	`POST /api`	`POST /api/users/{userId}/orders`
`GET /api/users/42/orders/7/items`	`GET /api`	`GET /api/users/{userId}/orders/{orderId}/items`
`GET /api/products`	`GET /api`	`GET /api/products`

Native OpenTelemetry solution

If you are using the native OpenTelemetry SDK (without ADOT), you can either override the span name using the transform processor in your OpenTelemetry Collector or directly in application code.

> **Note**
>
> You must configure your collector with an OTLP exporter to preserve the OpenTelemetry span name so that CloudWatch can parse it for the Application Signals operation name. For more information, see Sending OTLP data to CloudWatch.

Option 1 (recommended): Collector-side transformation

Use the transform processor to set span names based on URL patterns. The transform processor uses OTTL (OpenTelemetry Transformation Language) and can modify the span name field directly.

Order rules from shallowest to deepest – statements run sequentially, so more specific matches later in the list override the general ones set earlier. In the following example, the value of the span attribute `url.path` is matched and the resulting span name is set to the request method followed by the desired URL pattern.

```
processors:
  transform/operation_names:
    trace_statements:
      - context: span
        conditions:
          - IsMatch(attributes["url.path"], "^/api/contests(/|$)")
        statements:
          - set(name, Concat([attributes["http.request.method"], "/api/contests"], "
 "))

      - context: span
        conditions:
          - IsMatch(attributes["url.path"], "^/api/contests/[^/]+$")
        statements:
          - set(name, Concat([attributes["http.request.method"], "/api/contests/{id}"],
 " "))

      - context: span
        conditions:
          - IsMatch(attributes["url.path"], "^/api/contests/[^/]+/leaderboard(/|$)")
        statements:
          - set(name, Concat([attributes["http.request.method"], "/api/contests/{id}/
leaderboard"], " "))

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [resourcedetection, transform/operation_names, batch]
      exporters: [otlphttp/xray]
```

Option 2: Setting span name in application code

You can manually set the span name for server spans in your application code using the OpenTelemetry API. Set the name to `{HTTP_METHOD} {route_template}` where the route template uses parameterized placeholders. Note that this is a hardcoded fallback option, and you must manually update the span name in each HTTP request handler to apply this change.

Java

```
import io.opentelemetry.api.trace.Span;

// Inside your request handler
Span.current().updateName("GET /api/contests/{id}/leaderboard");
```

Python

```
from opentelemetry import trace

# Inside your request handler
span = trace.get_current_span()
span.update_name("GET /api/contests/{id}/leaderboard")
```

Go

```
import "go.opentelemetry.io/otel/trace"

// Inside your request handler
span := trace.SpanFromContext(ctx)
span.SetName("GET /api/contests/{id}/leaderboard")
```

Node.js

```
import { trace } from '@opentelemetry/api';

// Inside your request handler
const span = trace.getActiveSpan();
if (span) {
  span.updateName('GET /api/contests/{id}/leaderboard');
}
```

Manage high-cardinality operations

Application Signals includes settings in the CloudWatch agent that you can use to manage the cardinality of your operations and manage the metric exportation to optimize costs. By default, the metric limiting function becomes active when the number of distinct operations for a service over time exceeds the default threshold of 500. You can tune the behavior by adjusting the configuration settings.

Determine if metric limiting is activated

You can use the following methods to find if the default metric limiting is happening. If it is, you should consider optimizing the cardinality control by following the steps in the next section.

- In the CloudWatch console, choose **Application Signals**, **Services**. If you see an **Operation** named **AllOtherOperations** or a **RemoteOperation** named **AllOtherRemoteOperations**, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherOperations` for their `Operation` dimension, then metric limiting is happening.
- If any metrics collected by Application Signals have the value `AllOtherRemoteOperations` for their `RemoteOperation` dimension, then metric limiting is happening.

Optimize cardinality control

To optimize your cardinality control, you can do the following:

- Create custom rules to aggregate operations.
- Configure your metric limiting policy.

Create custom rules to aggregate operations

High-cardinality operations can sometimes be caused by inappropriate unique values extracted from the context. For example, sending out HTTP/S requests that include user IDs or session IDs in the path can lead to hundreds of disparate operations. To resolve such issues, we recommend that you configure the CloudWatch agent with customization rules to rewrite these operations.

In cases where there is a surge in generating numerous different metrics through individual `RemoteOperation` calls, such as `PUT /api/customer/owners/123`, `PUT /api/customer/owners/456`, and similar requests, Application Signals recommends that you consolidate these operations into a single `RemoteOperation`. One approach is to standardize all `RemoteOperation` calls that start with `PUT /api/customer/owners/` to a uniform format, specifically `PUT /api/customer/owners/{ownerId}`. The following example illustrates this. For information about other customization rules, see Enable CloudWatch Application Signals.

```
{
   "logs":{
      "metrics_collected":{
```

```
         "application_signals":{
            "rules":[
               {
                  "selectors":[
                     {
                        "dimension":"RemoteOperation",
                        "match":"PUT /api/customer/owners/*"
                     }
                  ],
                  "replacements":[
                     {
                        "target_dimension":"RemoteOperation",
                        "value":"PUT /api/customer/owners/{ownerId}"
                     }
                  ],
                  "action":"replace"
               }
            ]
         }
      }
   }
}
```

In other cases, high-cardinality metrics might have been aggregated to `AllOtherRemoteOperations`, and it might be unclear what specific metrics are included. The CloudWatch agent is able to log the dropped operations. To identify dropped operations, use the configuration in the following example to activate logging until the problem resurfaces. Then inspect the CloudWatch agent logs (accessible by container `stdout` or EC2 log files) and search for the keyword `drop metric data`.

```
{
  "agent": {
    "config": {
      "agent": {
        "debug": true
      },
      "traces": {
        "traces_collected": {
          "application_signals": {
          }
        }
      },
```

```
      "logs": {
        "metrics_collected": {
          "application_signals": {
            "limiter": {
              "log_dropped_metrics": true
            }
          }
        }
      }
    }
  }
```

Create your metric limiting policy

If the default metric limiting configuration doesn't address the cardinality for your service, you can customize the metric limiter configuration. To do this, add a `limiter` section under the `logs/metrics_collected/application_signals` section in the CloudWatch Agent configuration file.

The following example lowers the threshold of metric limiting from 500 distinct metrics to 100.

```
{
  "logs": {
    "metrics_collected": {
      "application_signals": {
        "limiter": {
          "drop_threshold": 100
        }
      }
    }
  }
}
```

Monitor the operational health of your applications with Application Signals

Use Application Signals within the CloudWatch console to monitor and troubleshoot the operational health of your applications:

- **Monitor your application services** — As part of daily operational monitoring, use the Services page to see a summary of all your services. See services with the highest fault rate or latency, and see which services have unhealthy service level indicators (SLIs). Select a service to open the Service detail page and see detailed metrics, service operations, Synthetics canaries, and client requests. This can help you troubleshoot and identify the root cause of operational issues.
- **Inspect your application topology** — Use the Application Map to understand and monitor your application topology over time, including the relationships between clients, Synthetics canaries, services, and dependencies. Instantly see service level indicator (SLI) health and view key metrics such as call volume, fault rate, and latency. Drill down to see more detailed information in the Service detail page.

Explore an example scenario that demonstrates how these pages can be used to quickly troubleshoot an operational service health issue, from initial detection to identifying root cause.

How Application Signals enables operational health monitoring

After you enable your application for Application Signals, your application services, APIs, and their dependencies are automatically discovered and displayed in the **Services**, **Service detail**, and **Application Map** pages. Application Signals collects information from multiple sources to enable service discovery and operational health monitoring:

- AWS Distro for OpenTelemetry (ADOT) — As part of enabling Application Signals, OpenTelemetry Java and Python auto-instrumentation libraries are configured to emit metrics and traces that are collected by the CloudWatch agent. The metrics and traces are used to enable discovery of services, operations, dependencies, and other service information.
- Service-level objectives (SLOs) — After you create service level objectives for your services, the Services, Service detail, and Application Map pages display service level indicator (SLI) health. SLIs can monitor latency, availability, and other operational metrics.
- CloudWatch Synthetics canaries — When you configure X-Ray tracing on your canaries, calls to your services from your canary scripts are associated with your service and displayed within the Service detail page.
- CloudWatch Real user monitoring (RUM) — When X-Ray tracing is enabled on your CloudWatch RUM web client, requests to your services are automatically associated and displayed within the service detail page.
- AWS Service Catalog AppRegistry — Application Signals auto-discovers AWS resources within your account and allows you to group them into logical applications created in AppRegistry. The

application name displayed in the Services page is based on the underlying compute resource that your services are running on.

> **Note**
>
> Application Signals displays your services and operations based on metrics and traces emitted within the current time filter that you chose. (By default, this is the past three hours.) If there is no activity within the current time filter for a service, operation, dependency, Synthetics canary, or client page, it won't be displayed.
> Up to 1,000 services can be displayed. Discovery of your services and service topology might be delayed up to 10 minutes. Evaluation of your service level indicator (SLI) health might be delayed up to 15 minutes.

> **Note**
>
> Application Signals console currently only supports choosing a maximum of one day within the 30 days time range.

View overall service activity and operational health with the Services page

Use the Services page to see a list of your services that are enabled for Application Signals. You can also view operational metrics and quickly see which services have unhealthy service level indicators (SLIs). Drill down to look for performance anomalies as you identify the root cause of operational issues. To view this page, open the CloudWatch console and choose **Services** under the **Application Signals** section in the left navigation pane.

For un-instrumented services, the Service overview page displays limited information with prominent calls-to-action to enable Application Signals instrumentation.

Explore operational health metrics for your services

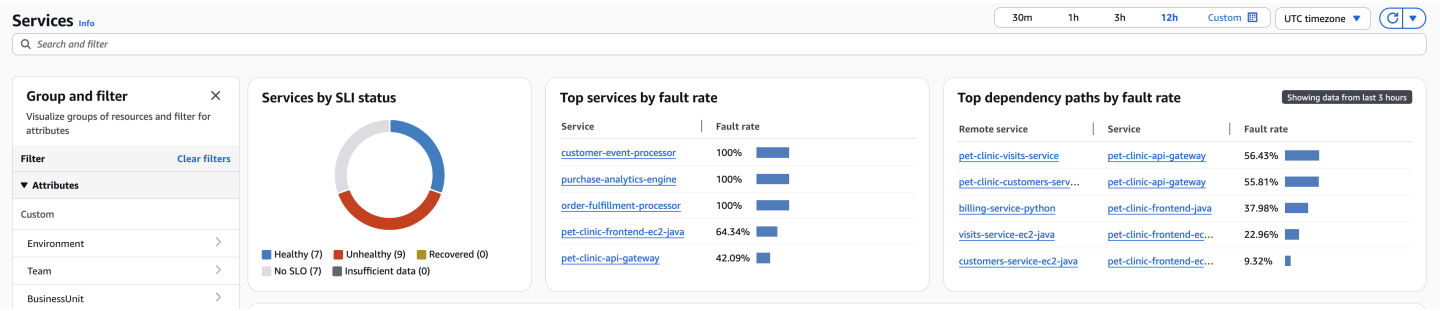
The top of the Services page includes an overall service operational health graph and several tables displaying top services and service dependencies by fault rate and list of services. The Services graph on the left displays a breakdown of the number of services that have healthy or unhealthy service level indicators (SLIs) during the current page-level time filter. SLIs can monitor latency, availability, and other operational metrics. View the top services by fault rate in the two

tables next to the graph. Select a service name in either table to open its service detail page page, which displays detailed service operation information. Select a dependency path to view service dependency details on its detail page.

Both tables display information for up to the past three hours, even if a longer time period filter is chosen at the top right of the page.

When using dynamic service grouping, the operational health metrics automatically aggregates data across all services within each group. This provides:

- Consolidated fault rates for service groups
- Group-level SLI health status
- Aggregated performance metrics that help identify problematic service clusters
- Quick identification of which groups require immediate attention during incidents



Monitor operational health with the Services table

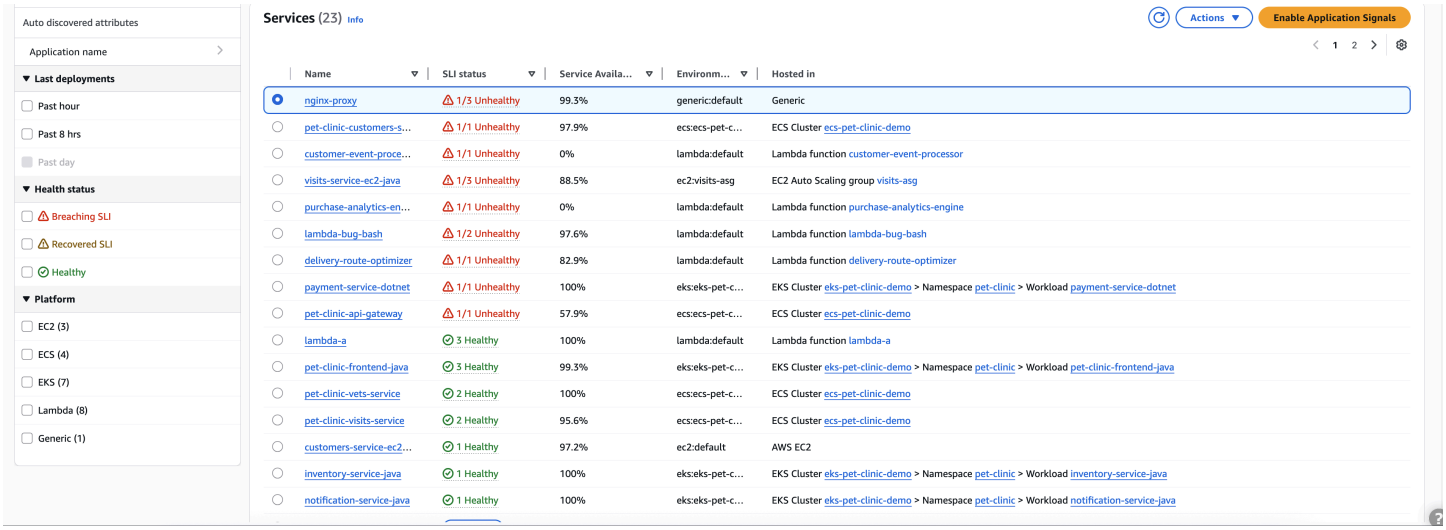
The Services table displays a list of your services that have been enabled for Application Signals. Choose **Enable Application Signals** to open a setup page and start configuring your services. For more information, see Enable Application Signals.

Filter the Services table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.

The advanced filtering options allows you to:

- Filter by service groups (both default and custom groupings)
- Filter by recent deployment activity

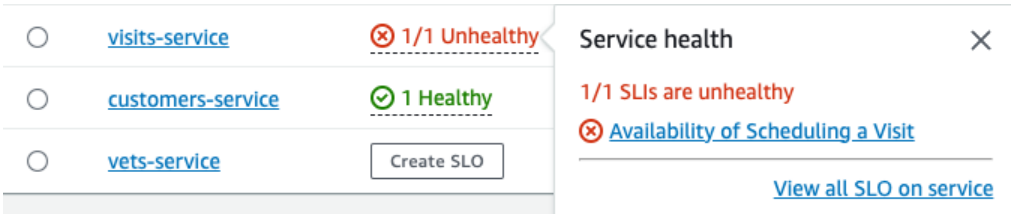
- Filter by Platform
- Filter by SLI Health
- Filter by Account ID (in cross-account observability setups)
- Filter by instrumentation status (instrumented vs un-instrumented)
- Filter by environment
- Filter by service health status



For un-instrumented services, the Service overview page displays limited information with prominent calls-to-action to enable Application Signals instrumentation. Un-instrumented services appear in the Services table even when they haven't been configured with Application Signals, helping you identify gaps in your observability coverage and prioritize which services to instrument next based on their position in your architecture.

Choose the name of any service in the table to view a service detail page containing service-level metrics, operations, and additional details. If you have associated the service's underlying compute resource with an application in AppRegistry or the Applications card on the AWS Management Console home page, choose the application name to display the application details in the myApplications console page. For services hosted in Amazon EKS, choose any link within the **Hosted in** column to view Cluster, Namespace, or Workload within CloudWatch Container Insights. For services running on Amazon ECS or Amazon EC2, the Environment value is shown.

Service level indicator (SLI) status is displayed for each service in the table. Choose the SLI status for a service to display a pop-up containing a link to any unhealthy SLIs, and a link to see all SLOs for the service.



If no SLOs have been created for a service, choose the **Create SLO** button within the **SLI Status** column. To create additional SLOs for any service, select the option button next to the service name, and then choose **Create SLO** at the top-right of the table. When you create SLOs, you can see at a glance which of your services and operations are performing well and which are unhealthy. See service level objectives (SLOs) for more information.

Service overview

After you select a service from the Services table, the Service overview page opens. This page provides a comprehensive view of your service's operational health and performance metrics. The overview displays these summary metrics:

- Total operations
- Service dependencies
- Canary monitoring status
- RUM client data

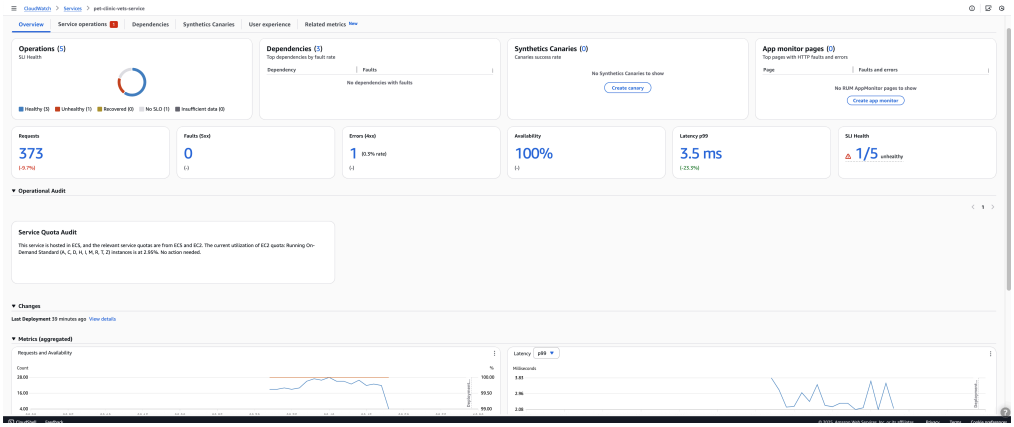
These metrics give you immediate insight into your service's current state.

You can visualize key operational performance indicators over time using a series of charts. To analyze trends and identify potential issues affecting your service health, adjust the time filter. All charts automatically update to reflect data for the selected time period.

The Audit findings section automatically detects and shows critical problems in your service's behavior, so you don't need to investigate manually. Application Signals analyzes your applications to report significant observations and potential problems, simplifying root cause analysis. These automated findings consolidate relevant traces, eliminating the need to navigate through multiple clicks. The audit system helps teams quickly identify issues and their underlying causes, enabling faster problem resolution.

You can use the Change events section to identify how recent deployments or configuration changes affect your service behavior. Application Signals automatically processes CloudTrail events

to track change events across your application. Monitor configuration and deployment events for services and their dependencies, providing immediate context for operational analysis and troubleshooting. Application Signals automatically correlates deployment times with performance changes, helping you quickly identify if recent deployments contributed to service issues.



View detailed service activity and operational health with the service detail page

When you instrument your application, Amazon CloudWatch Application Signals maps all of the services that your application discovers. Use the service detail page to see an overview of your services, operations, dependencies, canaries, and client requests for a single service. To view the service detail page, do the following:

- Open the CloudWatch console.
- Choose **Services** under the **Application Signals** section in the left navigation pane.
- Choose the name of any service from the **Services**, **Top services**, or dependency tables.

Under **schedule-visits**, you will see the account label and ID under the service name.

The service detail page is organized into the following tabs:

- Overview — Use this tab to see an overview of a single service, including the number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, top operations and dependencies. These metrics include time series data on latency, faults, and errors across all service operations for that service.
- Service operations — Use this tab to see a list of the operations that your service calls and interactive graphs with key metrics that measure the health of each operation. You can select a data point in a graph to obtain information about traces, logs, or metrics associated with that data point.

- Dependencies — Use this tab to see a list of dependencies that your service calls, and a list of metrics for those dependencies.
- Synthetics canaries — Use this tab to see a list of synthetics canaries that simulate user calls to your service, and key performance metrics for how those canaries.
- Client pages — Use this tab to see a list of client pages that call your service, and metrics that measure the quality of client interactions with your application.
- Related metrics — Use this tab to correlate related metrics, such as standard metrics, runtime metrics, and custom metrics for a service, it's operations or dependencies.

View your service overview

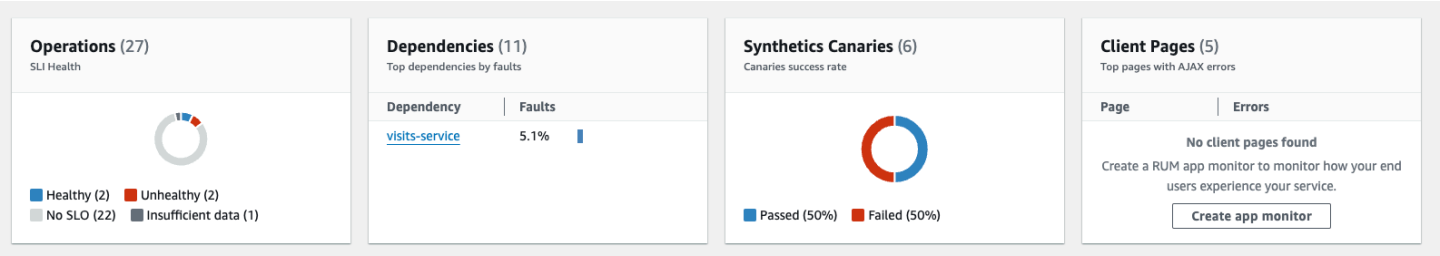
Use the service overview page to view a high-level summary of metrics for all service operations in a single location. Check the performance of all the operations, dependencies, client pages and synthetics canaries that interact with your application. Use this information to help you determine where to focus efforts to identify issues, troubleshoot errors, and find opportunities for optimization.

Choose any link in **Service Details** to view information that is related to a specific service. For example, for services hosted in Amazon EKS, the service details page shows **Cluster**, **Namespace**, and **Workload** information. For services hosted in Amazon ECS or Amazon EC2, the service details page shows the **Environment** value.

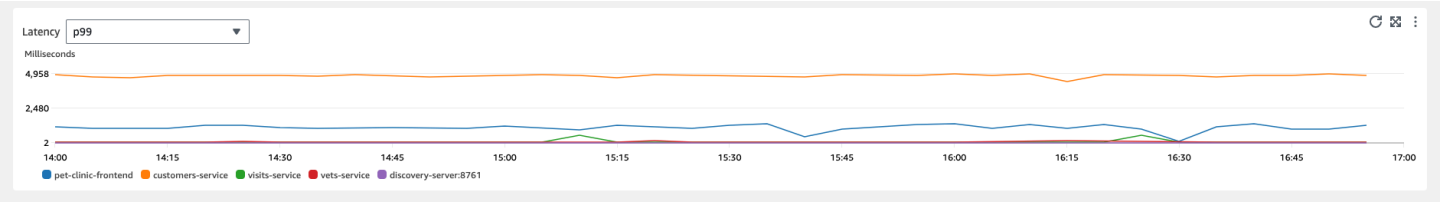
Under **Services**, the **Overview** tab displays a summary of the following:

- Operations – Use this tab to see the health of your service operations. The health status is determined by service level indicators (SLI) that are defined as a part of a service level objective (SLO).
- Dependencies – Use this tab to see the top dependencies of the services called by your application, listed by fault rate and to see the health of your service dependencies. The health status is determined by service level indicators (SLI) that are defined as a part of a service level objective (SLO).
- Synthetics canaries – Use this tab to see the result of simulated calls to endpoints or APIs associated with your service, and the number of failed canaries.
- Client pages – Use this tab to see top pages called by clients that have asynchronous JavaScript and XML (AJAX) errors.

The following illustration shows an overview of your services:

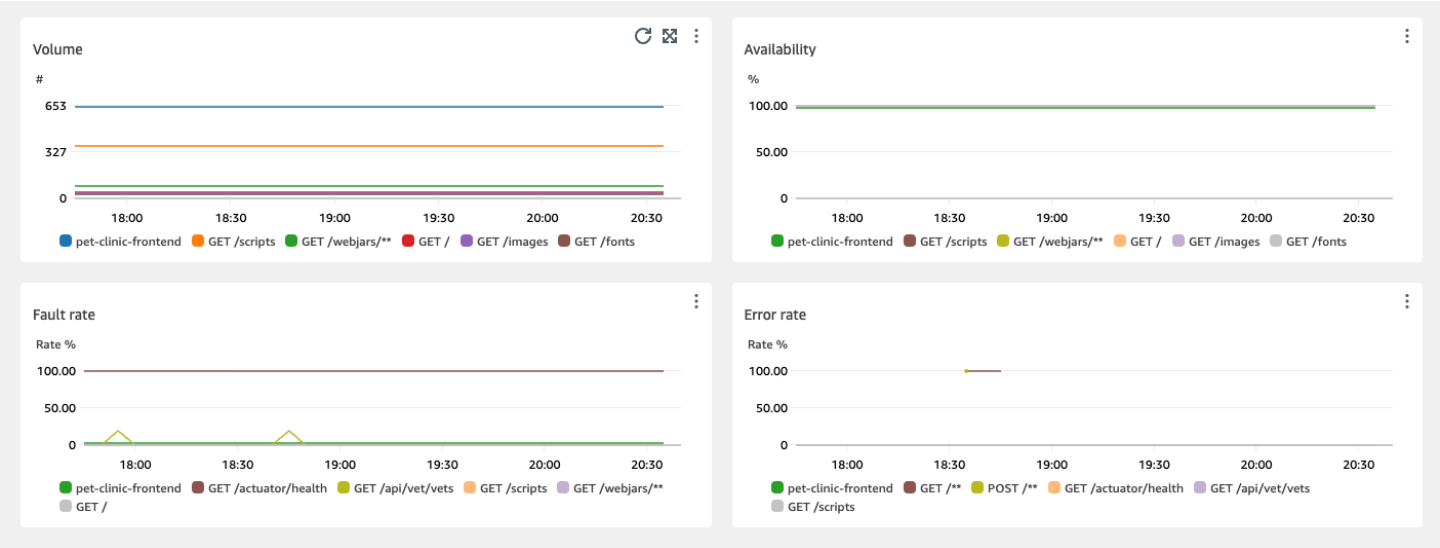


The **Overview** tab also displays a graph of dependencies with the highest latency across all services. Use the **p99**, **p90** and **p50** latency metrics to quickly assess which dependencies are contributing to your total service latency, as follows:



For example, the previous graph shows that 99% of the requests made to the **customer-service** dependency were completed in approximately 4,950 milliseconds. The other dependencies took less time.

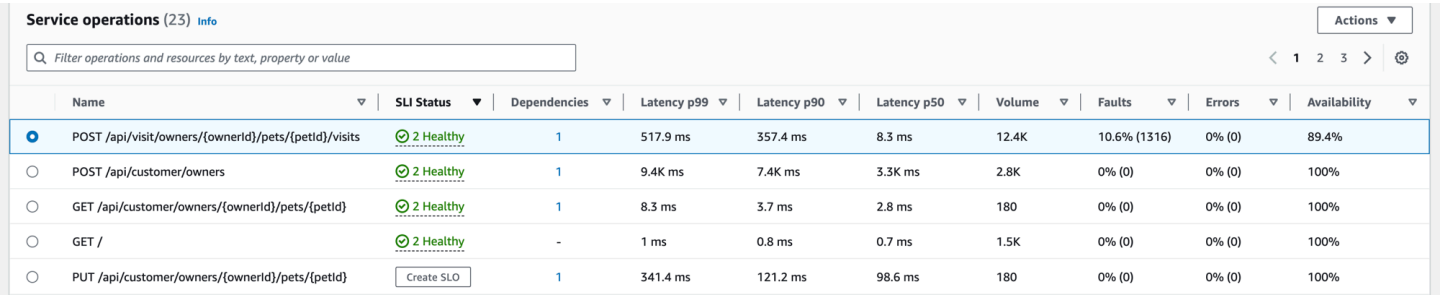
Graphs displaying the top four service operations by latency show the volume of requests, availability, fault rate, and error rate for those services, as shown in the following image:



The **Service details** section displays the details of the service including the **Account ID** and **Account label**.

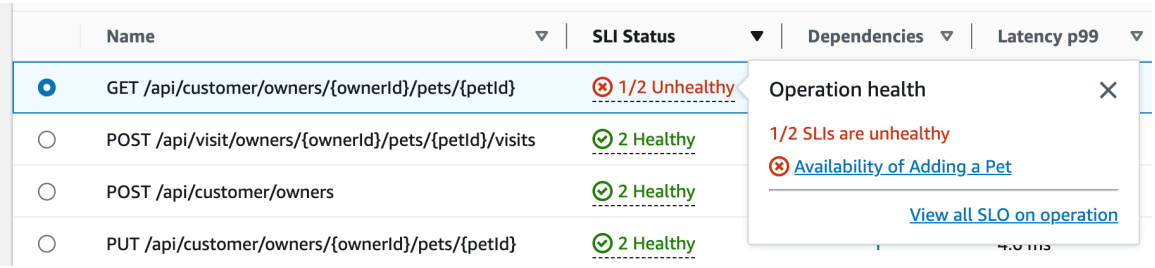
View your service operations

When you instrument your application, Application Signals discovers all of the service operations that your application calls. Use the **Service operations** tab to see a table that contains the service operations and a set of metrics that measure the performance of a selected operation. These metrics include SLI status, number of dependencies, latency, volume, faults, errors, and availability, as shown in the following image:



Filter the table to make it easier to find a service operation by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter.

Choose the SLI status for an operation to display a popup containing a link to any unhealthy SLI, and a link to see all SLOs for the operation, as shown in the following table:



The service operations table lists the SLI status, the number of healthy or unhealthy SLIs, and the total number of SLOs for each operation.

Use SLIs to monitor latency, availability, and other operational metrics that measure the operational health of a service. Use an SLO to check the performance and health status of your services and operations.

To create an SLO, do the following:

- If an operation does not have an SLO, choose the **Create SLO** button within the **SLI Status** column.

- If an operation already has an SLO, do the following:
  - Select the radio button next to the operation name.
  - Choose **Create SLO** from the **Actions** down arrow at the top right of the table.

For more information, see service level objectives (SLOs).

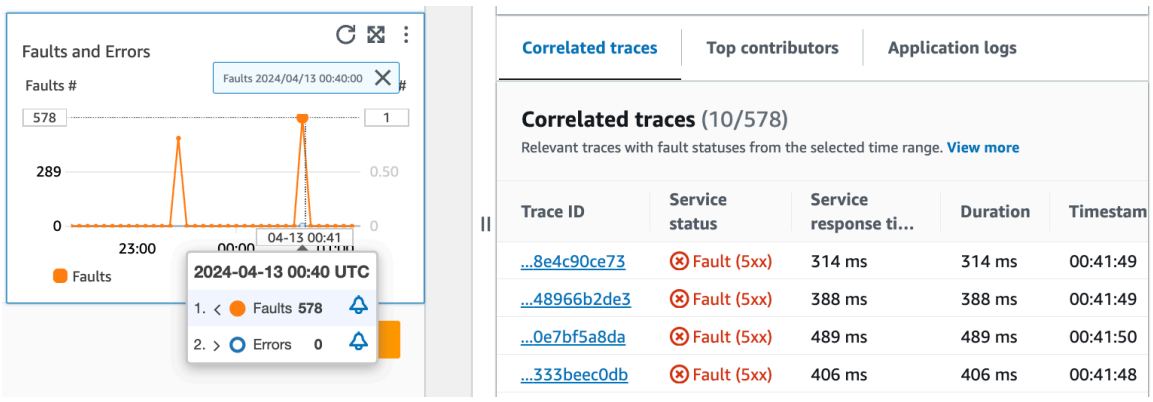
The **Dependencies** column shows the number of dependencies this operation calls. Choose this number to open the **Dependencies** tab filtered to the selected operation.

View service operations metrics, correlated traces, and application logs

Application Signals correlates service operation metrics with AWS X-Ray traces, CloudWatch Container Insights, and application logs. Use these metrics to troubleshoot operational health issues. To view metrics as graphical information, do the following:

1. Select a service operation in the **Service operations** table to see a set of graphs for the selected operation above the table with metrics for **Volume and Availability**, **Latency**, and **Faults and Errors**.
2. Hover over a point in a graph to view more information.
3. Select a point to open a diagnostic pane that shows correlated traces, metrics, and application logs for the selected point in the graph.

The following image shows the tooltip that appears after hovering over a point in the graph, and the diagnostic pane which appears after choosing on a point. The tooltip contains information about the associated data point in the **Faults and Errors** graph. The pane contains **Correlated traces**, **Top contributors**, and **Application logs** associated with the selected point.



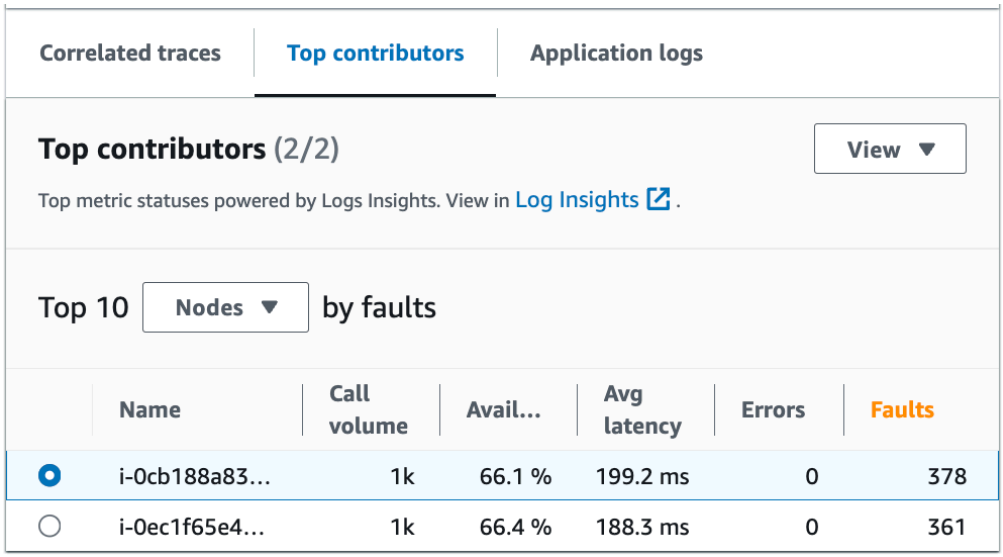
Correlated traces

Look at related traces to understand an underlying issue with a trace. You can check to see if correlated traces or any service nodes associated with them behave similarly. To examine correlated traces, choose a **Trace ID** from the **Correlated traces** table to open the X-Ray trace details page for the chosen trace. The trace details page contains a map of service nodes that are associated with the selected trace and a timeline of trace segments.

Top contributors

View the top contributors to find main input sources to a metric. Group contributors by different components to look for similarities within the group and understand how trace behavior differs between them.

The **Top contributors** tab gives metrics for **Call volume**, **Availability**, **Avg latency**, **Errors**, and **Faults** for each group. The following example image shows top contributors to a suite of metrics for an application deployed on an Amazon EKS platform:



The top contributors contains the following metrics:

- **Call volume** - Use the call volume to understand the number of requests per time interval for a group.
- **Availability** - Use availability to see what percentage of time that no faults were detected for a group.
- **Avg latency** - Use latency to check the average time that requests ran for a group over a time interval that depends on how long ago the requests that you are investigating were made. Requests that were made less than 15 days prior are evaluated over 1 minute intervals. Requests

that were made between 15 and 30 days prior, inclusive, are evaluated over 5 minute intervals. For example, if you are investigating requests that caused a fault 15 days ago, the call volume metric is equal to the number of requests per 5 minute interval.

- **Errors** - The number of errors per group measured over a time interval.
- **Faults** - The number of faults per group over a time interval.

Top contributors using Amazon EKS or Kubernetes

Use information about the top contributors for applications deployed on Amazon EKS or Kubernetesto see operational health metrics grouped by **Node**, **Pod** and **PodTemplateHash**. The following definitions apply:

- A **pod** is a group of one or more Docker containers that share storage and resources. A pod is the smallest unit that can be deployed on a Kubernetes platform. Group by pods to check if errors are related to pod-specific limitations.
- A **node** is a server that runs pods. Group by nodes to check if errors are related to node-specific limitations.
- A **pod template hash** is used to find a particular version of a deployment. Group by pod template hash to check if errors are related to a particular deployment.

Top contributors using Amazon EC2

Use information about the top contributors for applications deployed on Amazon EKS to see operational health metrics grouped by instance ID, and auto scaling group. The following definitions apply:

- An **Instance ID** is a unique identifier for the Amazon EC2 instance that your service runs. Group by instance ID to check if errors are related to a specific Amazon EC2 instance.
- An auto scaling group is a collection of Amazon EC2 instances that allow you to scale up or down the resources you need to serve your application requests. Group by auto scaling group if you want to check if errors are limited in scope to the instances inside the group.

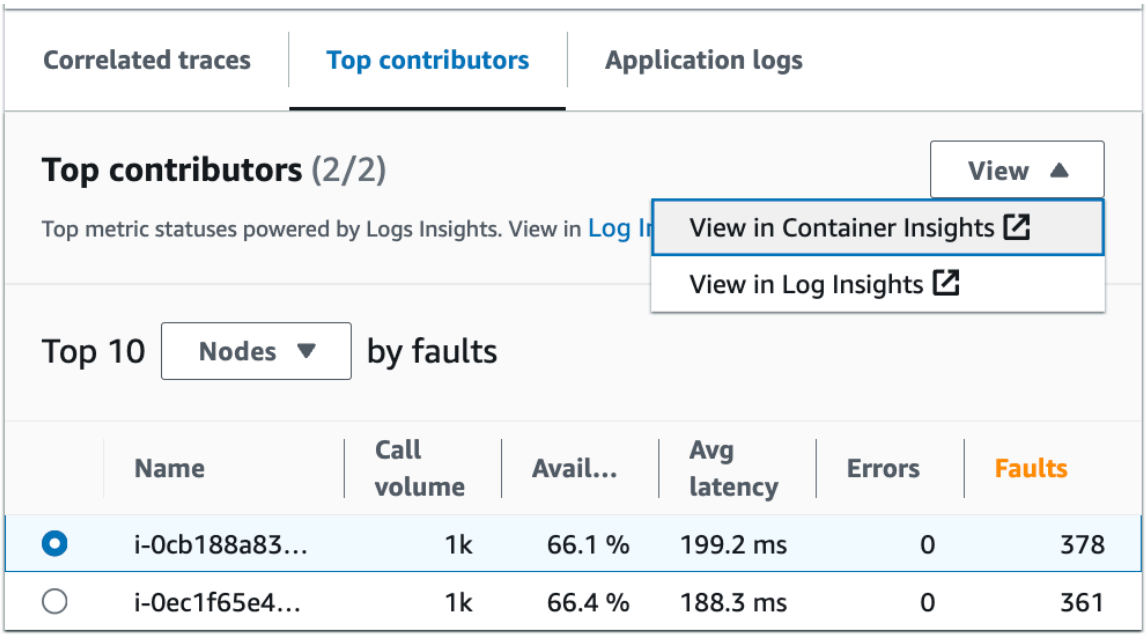
Top contributors using a custom platform

Use information about the top contributors for applications deployed using custom instrumentation to see operational health metrics grouped by **Host name**. The following definitions apply:

- A host name identifies a device such as an endpoint or Amazon EC2 instance that is connected to a network. Group by host name to check if your errors are related to a specific physical or virtual device.

View top contributors in Log Insights and Container Insights

View and modify the automatic query that generated metrics for your top contributors in Log Insights. View infrastructure performance metrics by specific groups such as pods or nodes in Container Insights. You can sort clusters, nodes or workloads by resource consumption and quickly identify anomalies or and mitigate risks pro-actively before end user experience is impacted. An image showing how to select these options follows:



In **Container Insights**, you can view metrics for your Amazon EKS or Amazon ECS container that are specific to the grouping of your top contributors. For example, if you grouped by pod for an EKS container to generate top contributors, container insights will show metrics and statistics filtered for your pod.

In **Log Insights**, you can modify the query that generated the metrics under **Top contributors** using the following steps:

1. Select **View in Log Insights**. The **Logs Insights** page that opens contains an query that is automatically generated and contains the following information:

   - The log cluster group name.

- The operation that you were investigating with CloudWatch.
- The aggregate of the operational health metric interacted with on the graph.

The log results are automatically filtered to show data from the last five minutes before you selected the data point on the service graph.

2. To edit the query, replace the generated text with your changes. You can also use the **Query generator** to help you generate a new query, or update the existing query.

Application logs

Use the query in the **Application logs** tab to generate logged information for your current log group, service and insert a timestamp. A log group is a group of log streams that you can define when you configure your application.

Use a log group to organize logs with similar characteristics including the following:

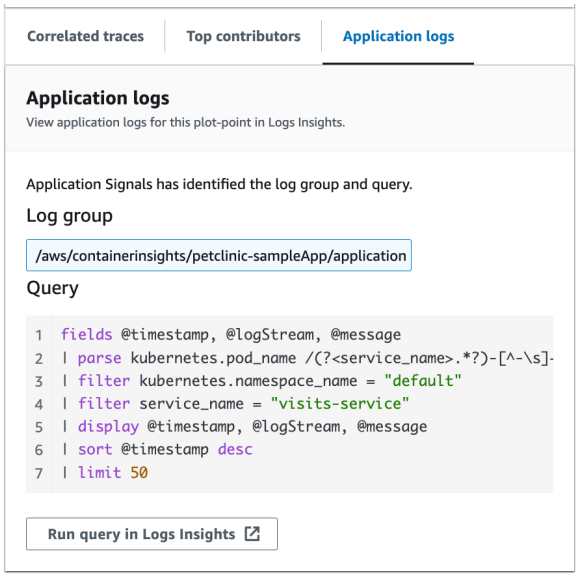
- Capture logs from a specific organization, source or function.
- Capture logs that are accessed by a particular user.
- Capture logs for a specific time period.

Use these log streams to track specific groups or time frames. You can also set up monitoring rules, alarms and notifications for these log groups. For more information about log groups, see Working with log groups and log streams.

The application logs query returns the logs, recurring text patterns and graphical visualizations for your log groups.

To run the query, select **Run query in Logs Insights** to either run the automatically generated query or modify the query. To edit the query, replace the automatically generated text with your changes. You can also use the **Query generator** to help you generate a new query or update the existing query.

The following image shows the sample query that is automatically generated based on the selected point in the service operations graph:



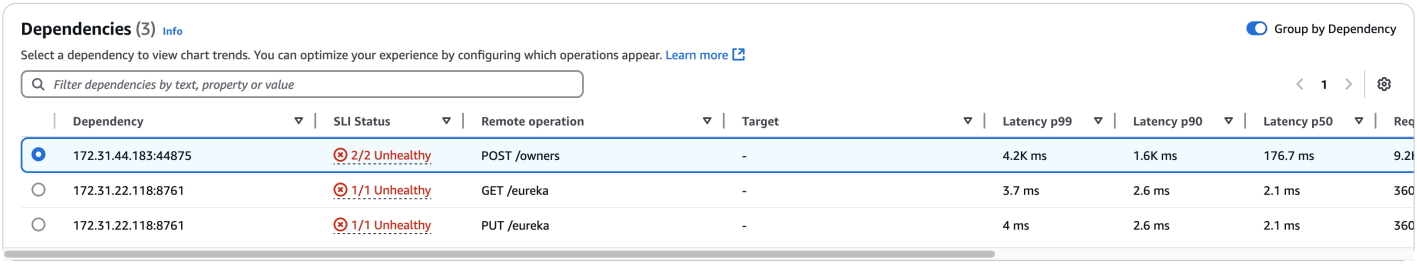
In the preceding image, CloudWatch has automatically detected the log group that is associated with your selected point, and included it in a generated query.

View your service dependencies

Choose the **Dependencies** tab to display the **Dependencies** table and a set of metrics for the dependencies of all service operations or a single operation. The table contains a list of dependencies discovered by Application Signals, including metrics for SLI status, latency, call volume, fault rate, error rate, and availability.

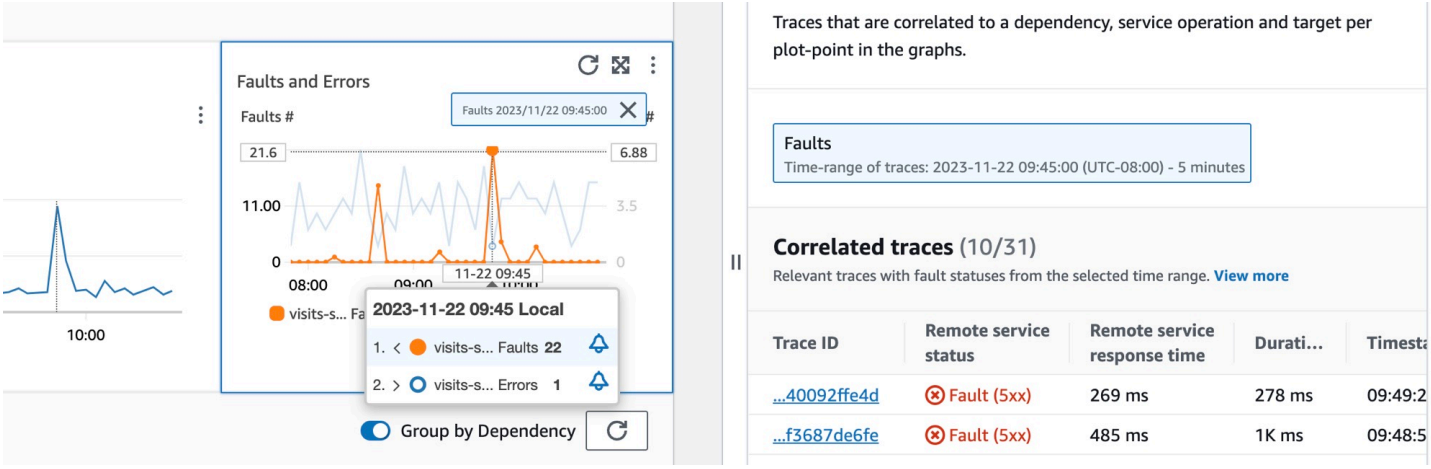
At the top of the page, choose an operation from the down arrow list to view its dependencies, or choose **All** to see dependencies for all operations.

Filter the table to make it easier to find what you're looking for, by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria and will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the table filter. Select **Group by Dependency** at the top right of the table to group dependencies by service and operation name. When grouping is turned on, expand or collapse a group of dependencies with the **+** icon next to the dependency name.



The **Dependency** column displays the dependency service name, while the **Remote Operation** column displays the service operation name. The **SLI status** column displays the number of healthy or unhealthy SLIs along with the total number of SLIs for each dependency. When calling AWS services, the **Target** column displays the AWS resource, such as DynamoDB table or Amazon SNS queue.

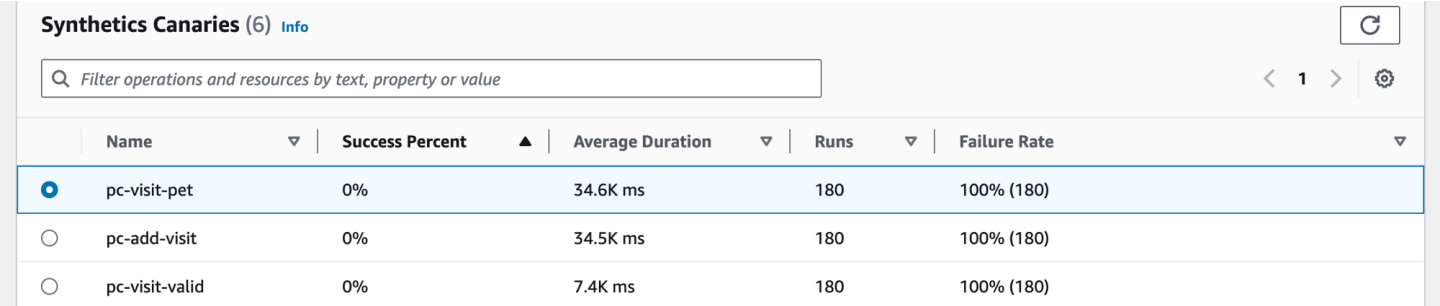
To select a dependency, select the option next to a dependency in the **Dependencies** table. This shows a set of graphs that display detailed metrics for call volume, availability, faults, and errors. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated traces for the selected point in the graph. Choose a trace ID from the **Correlated traces** table to open the X-Ray Trace details page for the selected trace.



View your Synthetics canaries

Choose the **Synthetics Canaries** tab to display the **Synthetics Canaries** table, and a set of metrics for each canary in the table. The table includes metrics for success percentage, average duration, runs, and failure rate. Only canaries that are enabled for AWS X-Ray tracing are displayed.

Use the filter text box in the synthetics canaries table to find the canary that you are interested in. Each filter that you create appears below the filter text box. Choose **Clear filters** at any time to remove the table filter.



Select the radio button next to the name of the canary to see a set of tabs containing graphs detailed metrics including success percentage, errors and duration. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows canary runs that correlate to the selected point. Select a canary run and choose the **Run time** to see artifacts for your selected canary run including logs, HTTP Archive (HAR) files, screenshots, and suggested steps to help you troubleshoot problems. Choose **Larn more** to open the CloudWatch Synthetics Canaries page next to **Canary runs**.

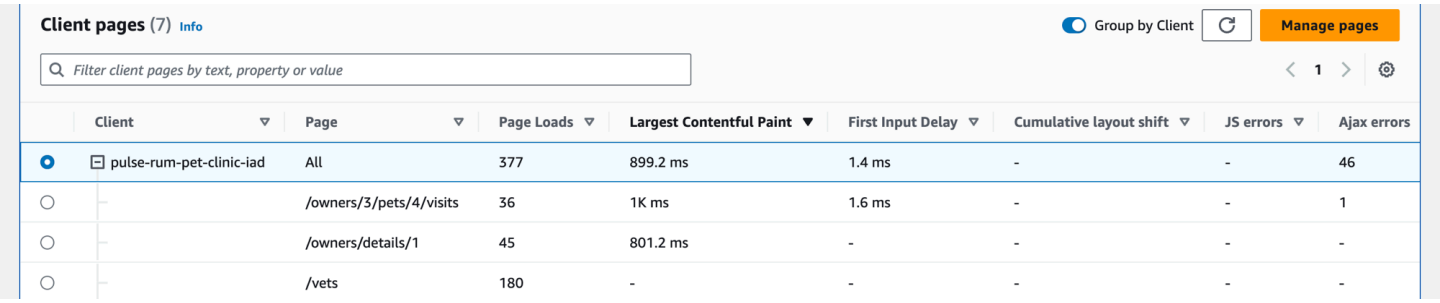


View your client pages

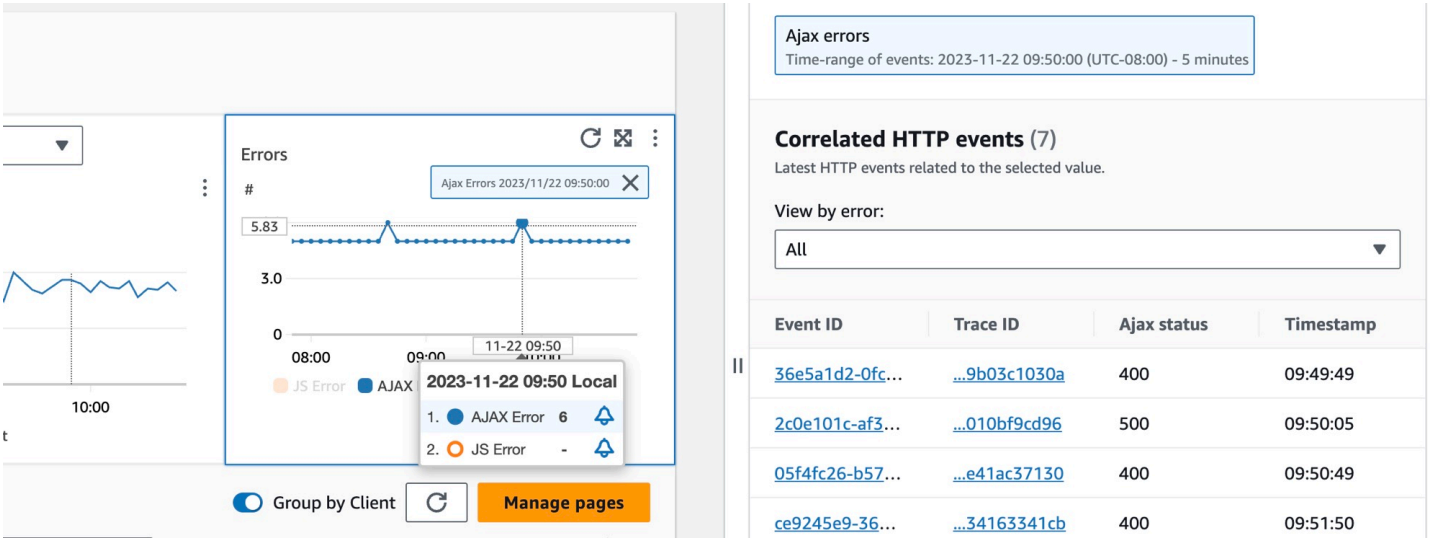
Choose the **Client pages** tab to display a list of client web pages that call your service. Use the set of metrics for the selected client page to measure the quality of your client's experience when interacting with a service or application. These metrics include page loads, web vitals, and errors.

To display your client pages in the table, you must configure your CloudWatch RUM web client for X-Ray tracing and turn on Application Signals metrics for your client pages. Choose **Manage pages** to select which pages are enabled for Application Signals metrics.

Use the filter text box to find the client page or application monitor that you are interested in below the filter text box. Choose **Clear filters** to remove the table filter. Select **Group by Client** to group client pages by client. When grouped, choose the **+** icon next to a client name to expand the row and see all pages for that client.



To select a client page, select the option next to a client page in the **Client pages** table. You will see a set of graphs that display detailed metrics. Hover over a point in a graph to see a popup containing more information. Select a point in a graph to open a diagnostic pane that shows correlated performance navigation events for the selected point in the graph. Choose an event ID from the list of navigation events to open the CloudWatch RUM Page view for the chosen event.



> **Note**
>
> To see AJAX errors within your client pages, use the CloudWatch RUM web client version 1.15 or newer.
>
> Up to 100 operations, canaries, and client pages, and up to 250 dependencies, can be shown per service.

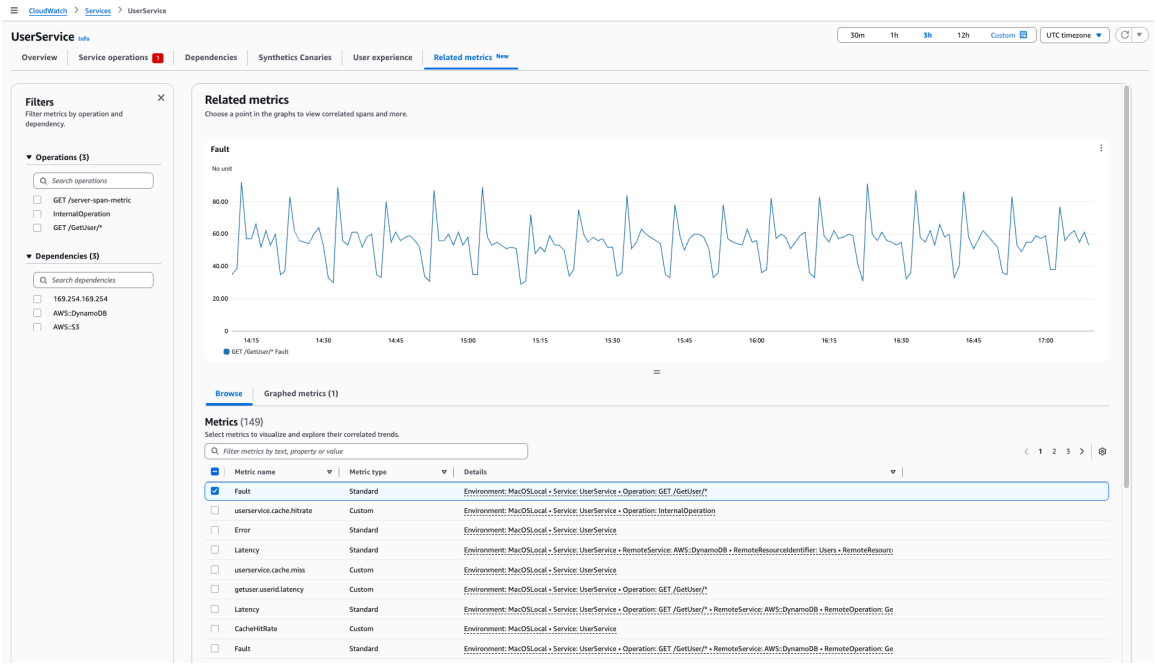
View Related metrics

Use the Related metrics tab to visualize multiple metrics, identify correlation patterns, and determine root causes of issues.

The metrics table shows three types of metrics:

- Standard metrics – Application Signals collects standard application metrics from the services that it discovers. For more information, see Standard application metrics collected
- Runtime metrics – Application Signals uses the AWS Distro for OpenTelemetry SDK to automatically collect OpenTelemetry-compatible metrics from your Java and Python applications. For more information, see Rumtime metrics
- Custom metrics – Application Signals enables you to generate custom metrics from your application. For more information, see Custom metrics with Application Signals

You can access the Related metrics tab from Service Overview, Service Operations, Dependencies, Synthetics canaries, or RUM tabs.



- The left navigation panel starts with all operations and dependencies unselected
- The graph initially shows the Fault metric from the operation with the highest fault rate

Before you begin correlation analysis, make sure you have data points visible in Service Operations or Dependencies. To analyze correlations:

1. Open the Service Operations or Dependencies page.
2. Select a data point on any graph.
3. In the right-panel, choose **Correlate with Other Metrics**.

4. On the **Related metrics** tab that opens, you'll see:

   - Your selected operation or dependency in the left navigation
   - Your selected metric graphed in the *Browse metrics* table
   - Correlated spans when you select a data point

To graph multiple metrics, select one or more metrics from the **Browse** view in the **Related metrics** tab. Choose **Graphed Metrics** to view all graphed metrics.

To filter metrics, use the left panel filters to focus on specific operations or dependencies and use the table header filter bar to search by name, type, or other attributes. These filtering options help you detect patterns and troubleshoot issues more efficiently.

To analyze related metrics in detail, select a data point in the **Related metrics** tab. You can then view:

- Top Contributors – Analyzes metrics by running CloudWatch Logs Insights queries. These queries process Enhanced Metrics Format (EMF) records that contain key attributes for detailed analysis for the following:
  - Latency measurements
  - Fault occurrences
  - Service availability metrics

  The following metrics do not support Top Contributors:
  - OTEL Metrics
  - Server-side Span Metrics

  You can view Top Contributors for RED Metrics and Client-side Span Metrics.
- Correlated Spans – The Correlated Spans section works consistently with the Service Operations tab. To help you identify related traces and metrics, the correlation mechanism works by:
  - Comparing metric names with span attributes
  - Identifying matching patterns during the selected time period
  - Displaying relevant trace information

  To effectively analyze your metrics and spans together, you need to understand how different metric types correlate. Here are the key limitations:

- OTEL Metrics don't correlate with spans because they use independent naming systems
- To correlate Server or Client-side Span Metrics with spans:
- Include a Service dimension field in your configuration
- Without this Service dimension, you cannot correlate these metrics with spans

- Log Applications – For information on log application, see Application logs

View your application topology and monitor operational health with the CloudWatch application map

> **Note**
>
> The CloudWatch application map replaces the Service Map. To see a map of your application based on AWS X-Ray traces, open the X-Ray Trace Map. Choose **Trace Map** under the **X-Ray** section in the left navigation pane of the CloudWatch console.

After enabling your application for Application Signals, the application map displays nodes representing your groups. You then drill down in these groups to view your services and their dependencies. Use the application map to view the topology of your application clients, synthetics canaries, services and dependencies, and monitor operational health. To view the application map, open the CloudWatch console and choose **Application Map** under the **Application Signals** section in the left navigation pane.

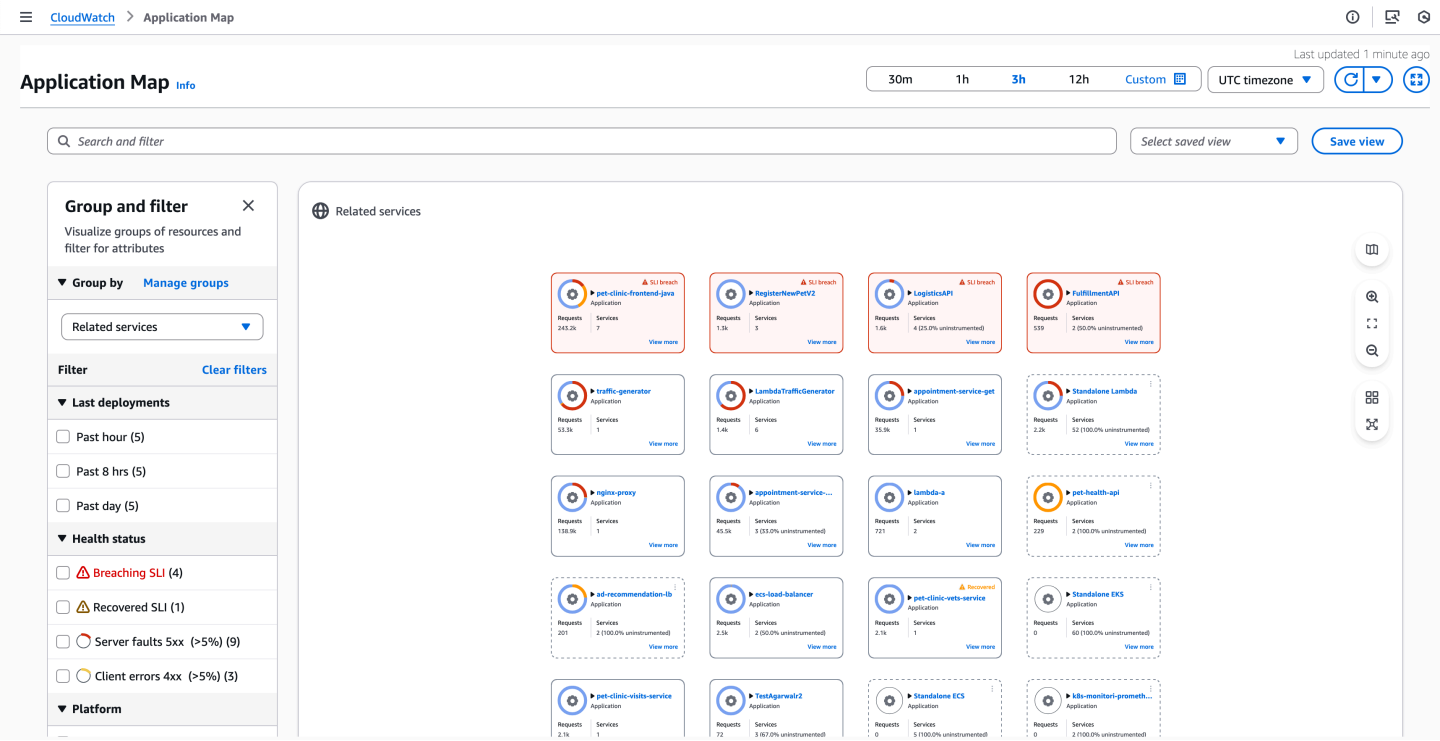
After you enable your application for Application Signals, use the application map to make it easier to monitor your application's operational health:

- View connections between client, canary, service, and dependency nodes to help you understand your application topology and execution flow. This is especially helpful if your service operators are not your development team.
- See which services are meeting or not meeting your service level objectives (SLOs). When a service is not meeting your SLOs, you can quickly identify whether a downstream service or dependency might be contributing to the issue or impacting multiple upstream services.
- Select an individual client, synthetics canary, service, or dependency node to see related metrics. The Service details page shows more detailed information about operations, dependencies, synthetics canaries, and client pages.

- Filter and zoom the application map to make it easier to focus on a part of your application topology, or see the entire map. Create a filter by choosing one or more properties from the filter text box. As you choose each property, you are guided through filter criteria. You will see the complete filter below the filter text box. Choose **Clear filters** at any time to remove the filter.
- Monitor services across multiple AWS accounts in a single unified application map. Services from different accounts are clearly identified with account information, enabling unified observability for distributed applications.
- Identify services not yet instrumented in your application. Application Signals automatically detects and displays services that haven't been instrumented yet, helping you achieve complete observability coverage. Un-instrumented services are visually distinguished on the map to help you prioritize instrumentation efforts.
- Group and filter services to create customized views that match your workflows. This organization helps you quickly find and access the services you use most frequently
- Save your filtered and grouped views to quickly return to frequently used configurations

Explore the application map

When you visit the application map, by default it shows services grouped by **Related services**. Related services group services based on their dependencies. For example, if Service A calls Service B, which calls Service C, they're grouped under Service A. You can view SLI health, metrics and service count for all services in each group.



Choose a tab for information about exploring each kind of node and the edges (connections) between them.

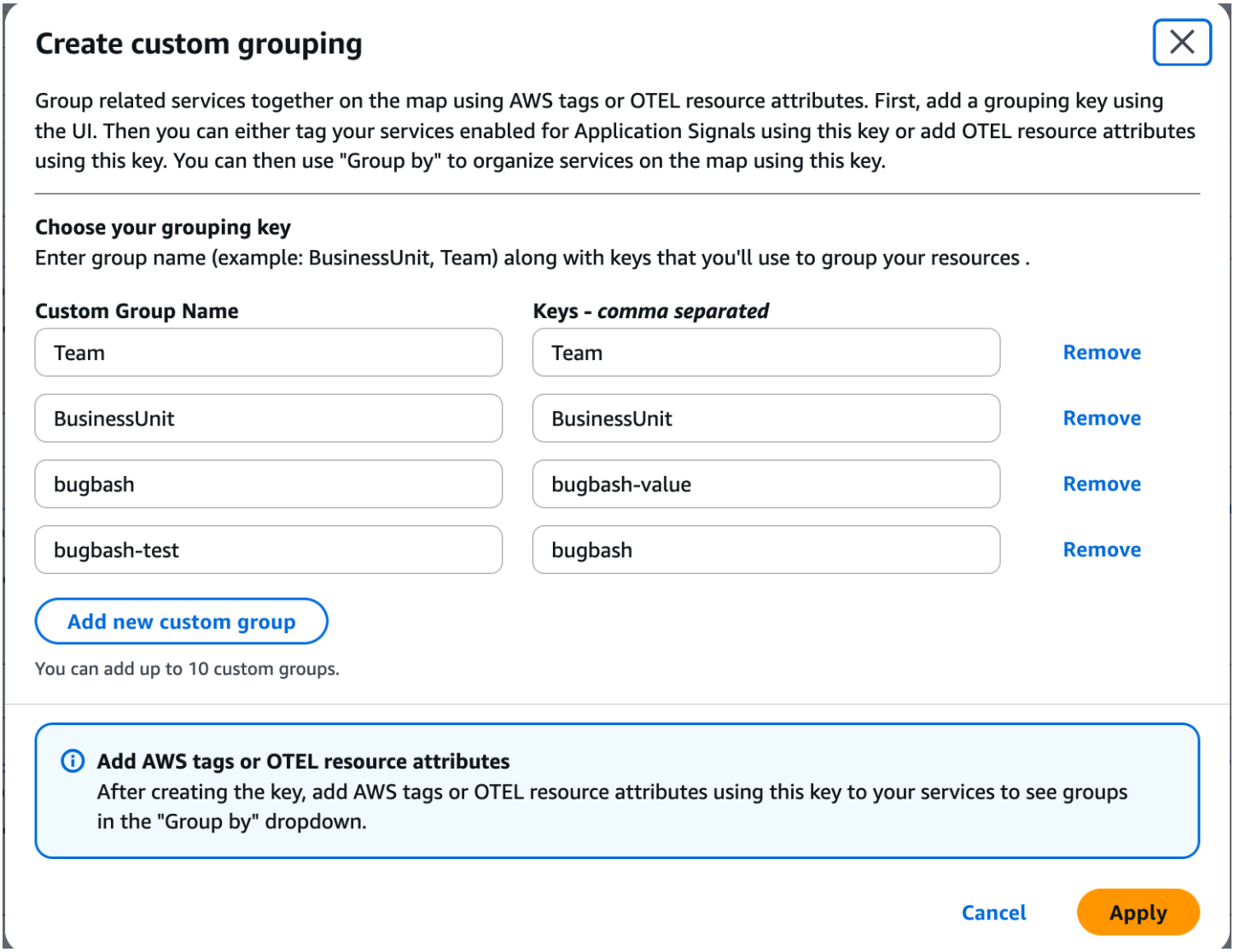
Dynamic grouping and filtering

You can click the **Group by** dropdown to use different grouping options. By default, Application Map provides 2 groupings:

- **Related services** - Groups services based on their dependencies
- **Environment** - Groups services by their environment

If you want to define your own custom grouping, click **Manage groups** to define custom groups and then tag your services or add OTEL Resource Attributes with the group key.

> **Note**
>
> To enable grouping through OTEL resource attributes, the CloudWatch agent version must be v1.300056.0 or later.

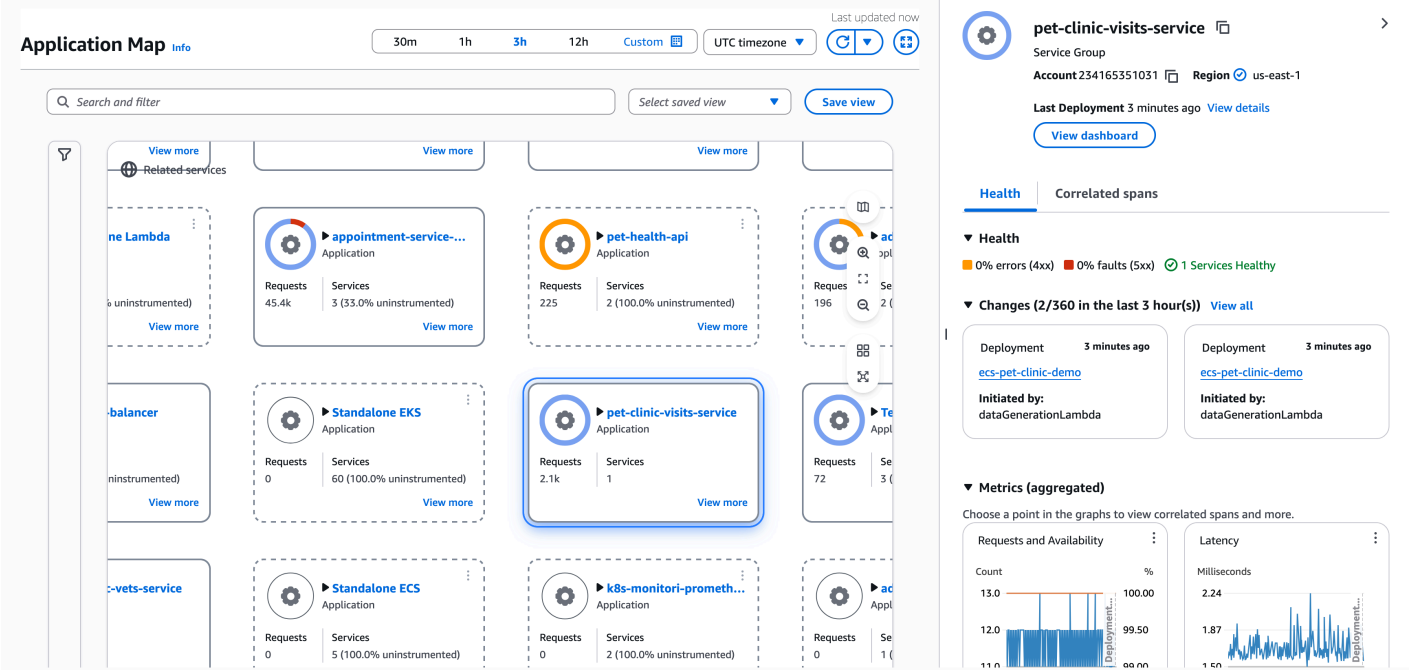


Default grouping in Application Signals automatically organizes services based on their downstream dependencies. The system analyzes the service dependency graph and creates groups where the root node (a service with no upstream dependencies) becomes the group name. All services that depend on this root service, either directly or indirectly, are automatically included in the group. For example, if Service A calls Service B, which in turn calls Service C, all three services will be grouped together with Service A as the group name since it's the root of the dependency chain. This automatic grouping mechanism provides a natural way to visualize and manage related services based on their actual runtime interactions and dependencies.

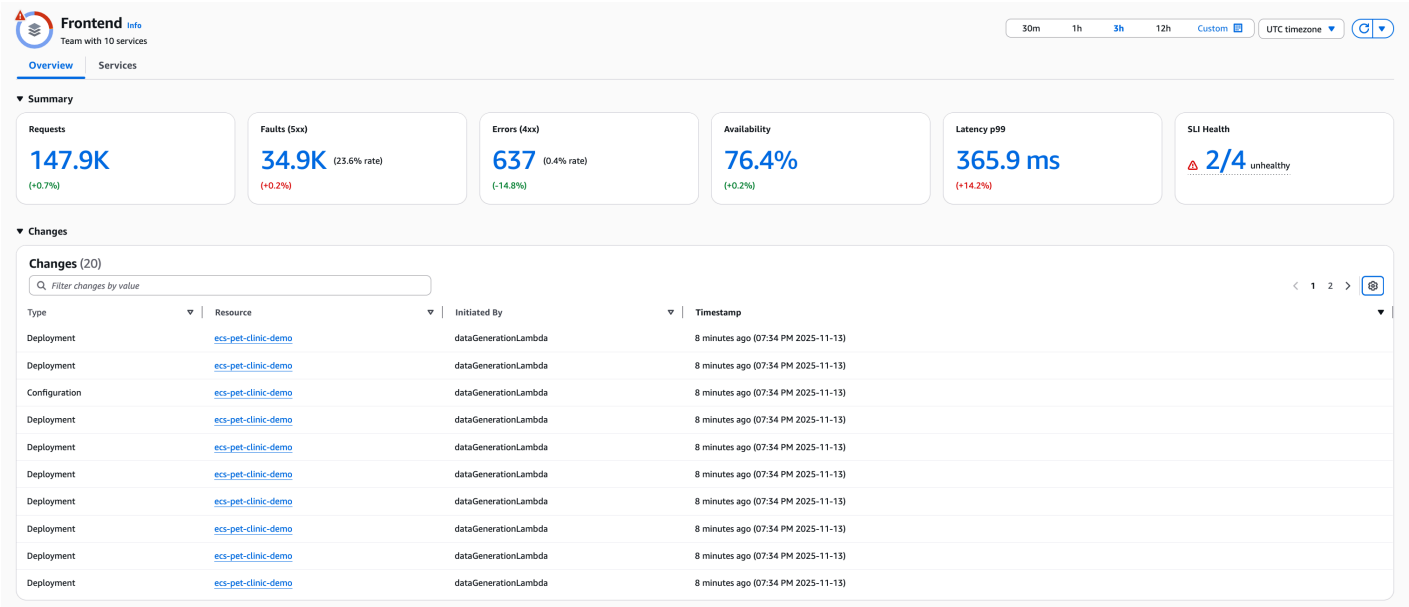
Group actions and insights

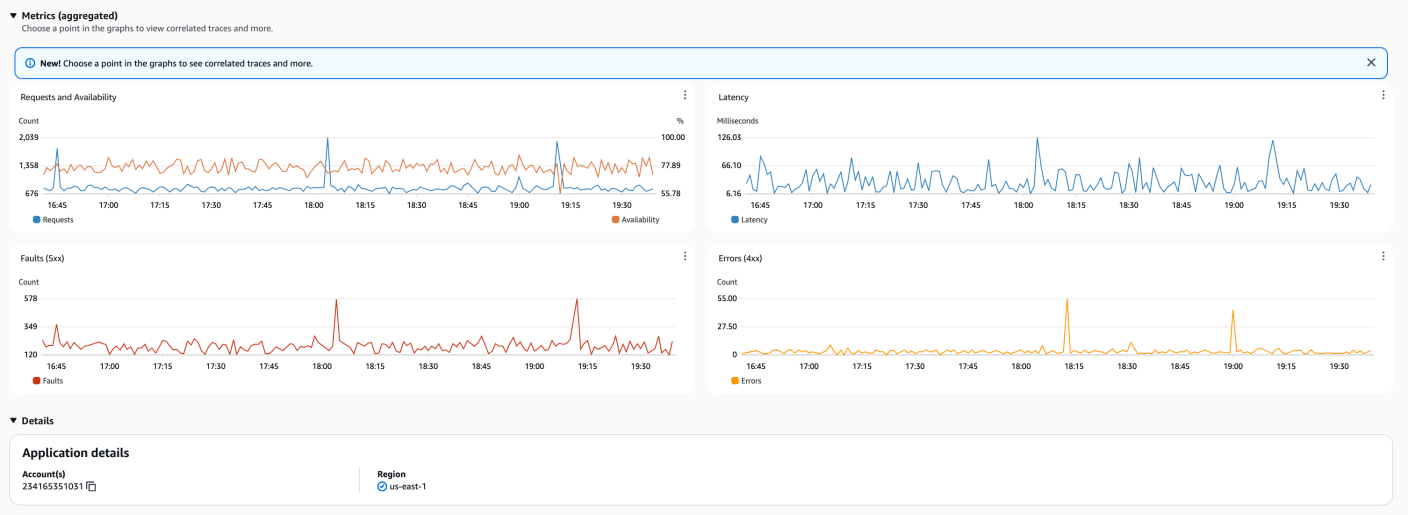
For each group, you can perform the following actions:

- Click **View more** to view metrics charts, the last two change events, and last deployment time for the group

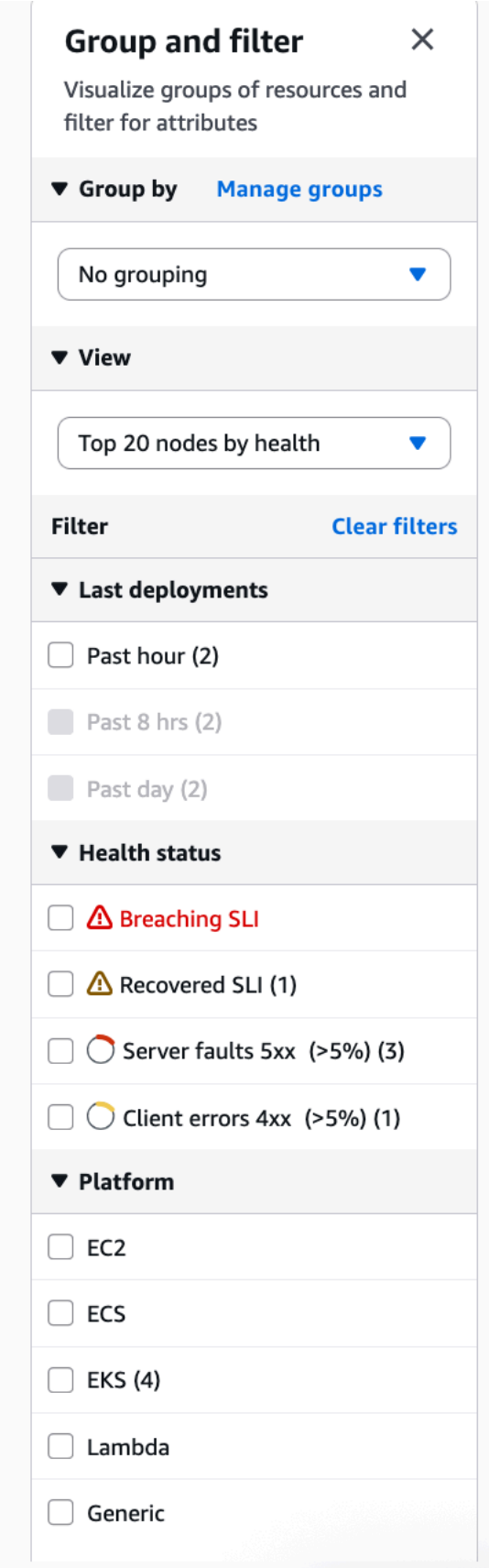


- Click **View dashboard** to view metrics dashboard, change events table, and service list for the group

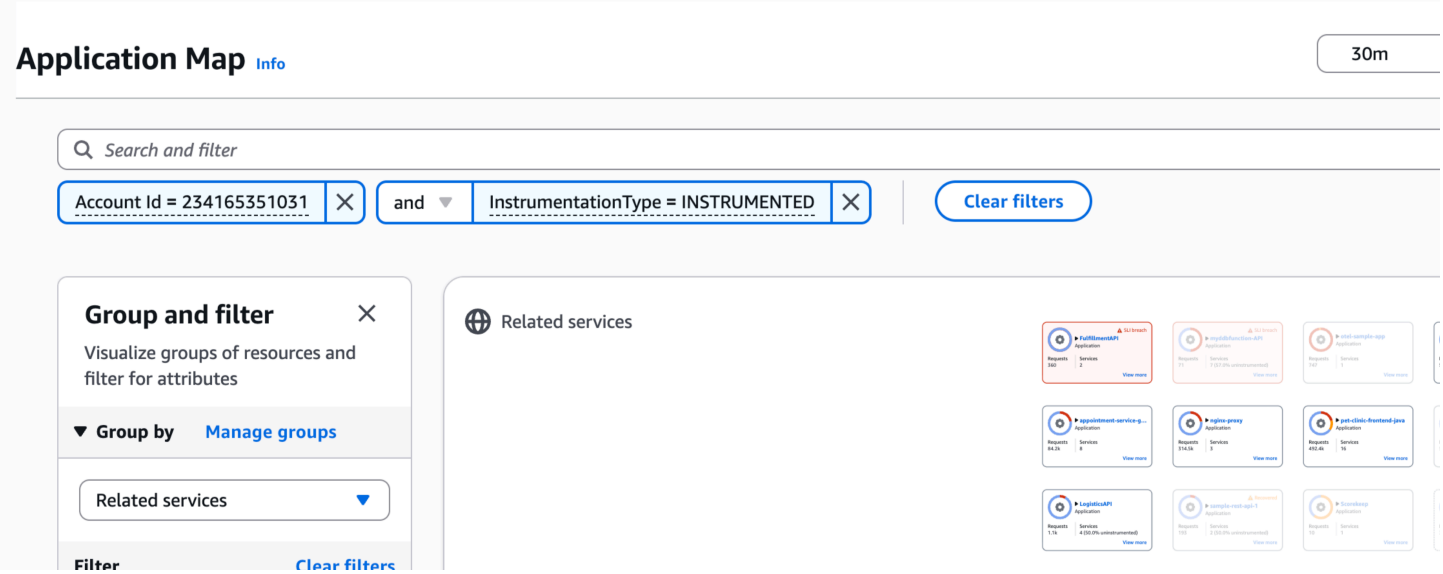




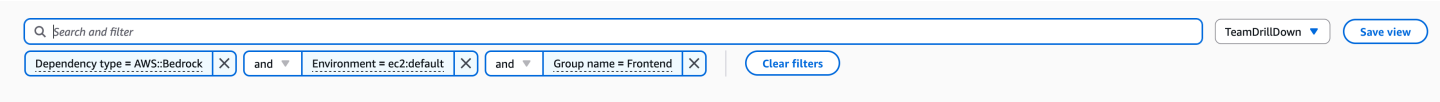
You can use **Group and filter** on the left bar to filter groups which have services with deployment time, SLI health status or compute platform type.


Group and filter
Visualize groups of resources and filter for attributes
Group by
Manage groups
No grouping
View
Top 20 nodes by health
Filter
Clear filters
Last deployments
Past hour (2)
Past 8 hrs (2)
Past day (2)
Health status
Breaching SLI
Recovered SLI (1)
Server faults 5xx (>5%) (3)
Client errors 4xx (>5%) (1)
Platform
EC2
ECS
EKS (4)
Lambda
Generic

You can also filter by account to view services from specific AWS accounts in your cross-account observability setup.



Use the **Search and filter** bar to search groups by name or search groups which contain specific service environment or dependency. Filter by account ID to focus on services from specific accounts.



Configuring custom groups

Custom grouping allows you to organize your services logically based on your business requirements and operational priorities. This feature enables you to view and save defined views prioritized by your specific needs, create groups based on team ownership, and assemble groups of services needed for critical business transactions.

Create the custom group names (the group names you will see in the UI) and the corresponding group key names. Complete this step either from the Application Signals UI or using the PutGroupingConfiguration API.

Group key names can be either, AWS tag key or OTEL resource attribute for your service. When deciding between tags and OTEL resource attributes, consider your compute platform:

- For single-service platforms (for example, Lambda or Auto Scaling Group) – Use AWS tags
- For multi-service platforms (for example, Amazon EKS cluster) – Use OTEL resource attributes for more granular grouping

Adding AWS tags

Add an AWS tag with the custom group key as a key and a value to an Amazon EKS cluster. When there are multiple services running in one Amazon EKS cluster all of them are tagged with the same custom group key. For example, when Amazon EKS Cluster A has Service 1, Service 2 and Service 3 running, adding an AWS tag with key *Team X* to the cluster will add all three services to *Team X*. To add only specific services to *Team X*, add OTEL resource attributes for the services as shown below.

Adding OTEL resource attributes

To add an OTEL resource attribute, see the configuration below:

General configuration

Configure the `OTEL_RESOURCE_ATTRIBUTES` environment variable in your application using the custom group key-value pairs. The keys are listed under `aws.application_signals.metric_resource_keys` separated by &.

For example, to create custom groups using `Application=PetClinic` and `Owner=Test`, use the following:

```
OTEL_RESOURCE_ATTRIBUTES=Application=PetClinic,Owner=Test,aws.application_signals.metric_resour
```

Platform-specific configuration

The following are the deployment specifications.

Amazon EKS and native kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  replicas: 1
  ...
  template:
    spec:
      containers:
      - name: your-app
        image: your-app-image
```

```
        env:
          ...
          - name: OTEL_RESOURCE_ATTRIBUTES
            value:
 Application=PetClinic,Owner=Test,aws.application_signals.metric_resource_keys=Application&Owne
```

Amazon EC2

Add OTEL_RESOURCE_ATTRIBUTES to your application start script. For the complete example, see Adding OTEL_RESOURCE_ATTRIBUTES.

```
...
OTEL_RESOURCE_ATTRIBUTES="service.name=
$YOUR_SVC_NAME,Application=PetClinic,Owner=Test,aws.application_signals.metric_resource_keys=Ap
 \
java -jar $MY_JAVA_APP.jar
```

Amazon ECS

Add OTEL_RESOURCE_ATTRIBUTES to the TaskDefinition. For the complete example, see Enable on Amazon ECS.

```
{
  "name": "my-app",
   ...
  "environment": [
    {
      "name": "OTEL_RESOURCE_ATTRIBUTES",
      "value": "service.name=
$YOUR_SVC_NAME,Application=PetClinic,Owner=Test,aws.application_signals.metric_resource_keys=Ap
 portalOwner"
    },
    ...
  ]
}
```

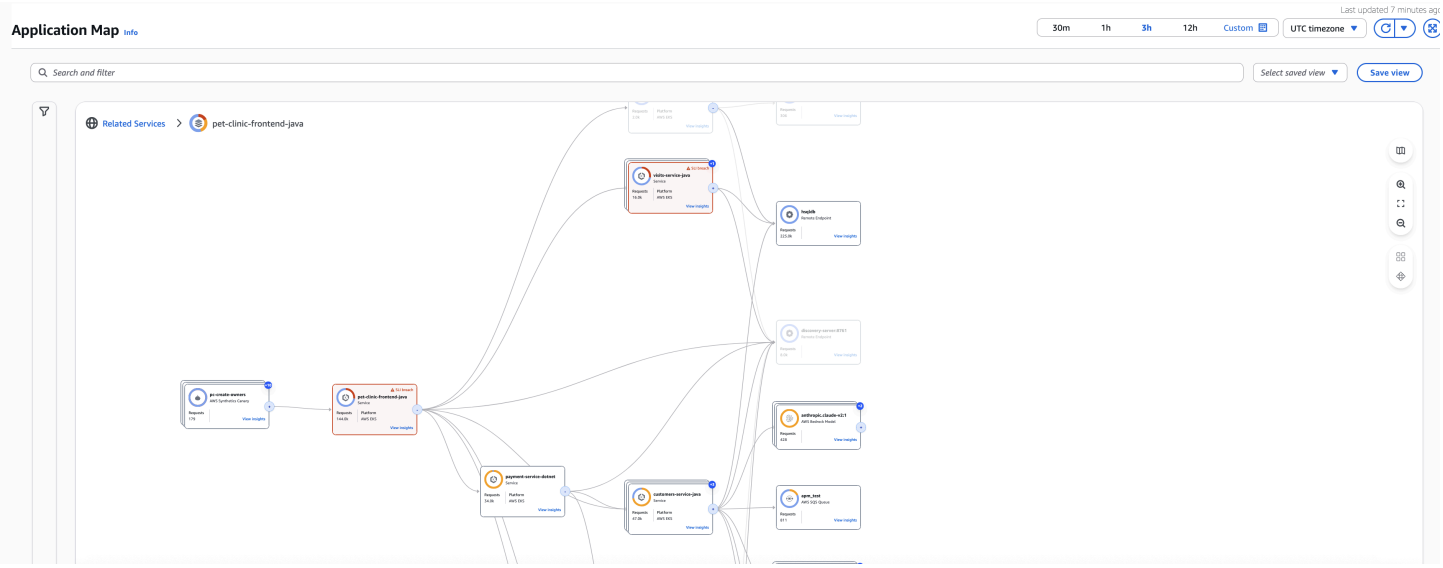
Lambda

Add OTEL_RESOURCE_ATTRIBUTES to the Lambda environment variable.

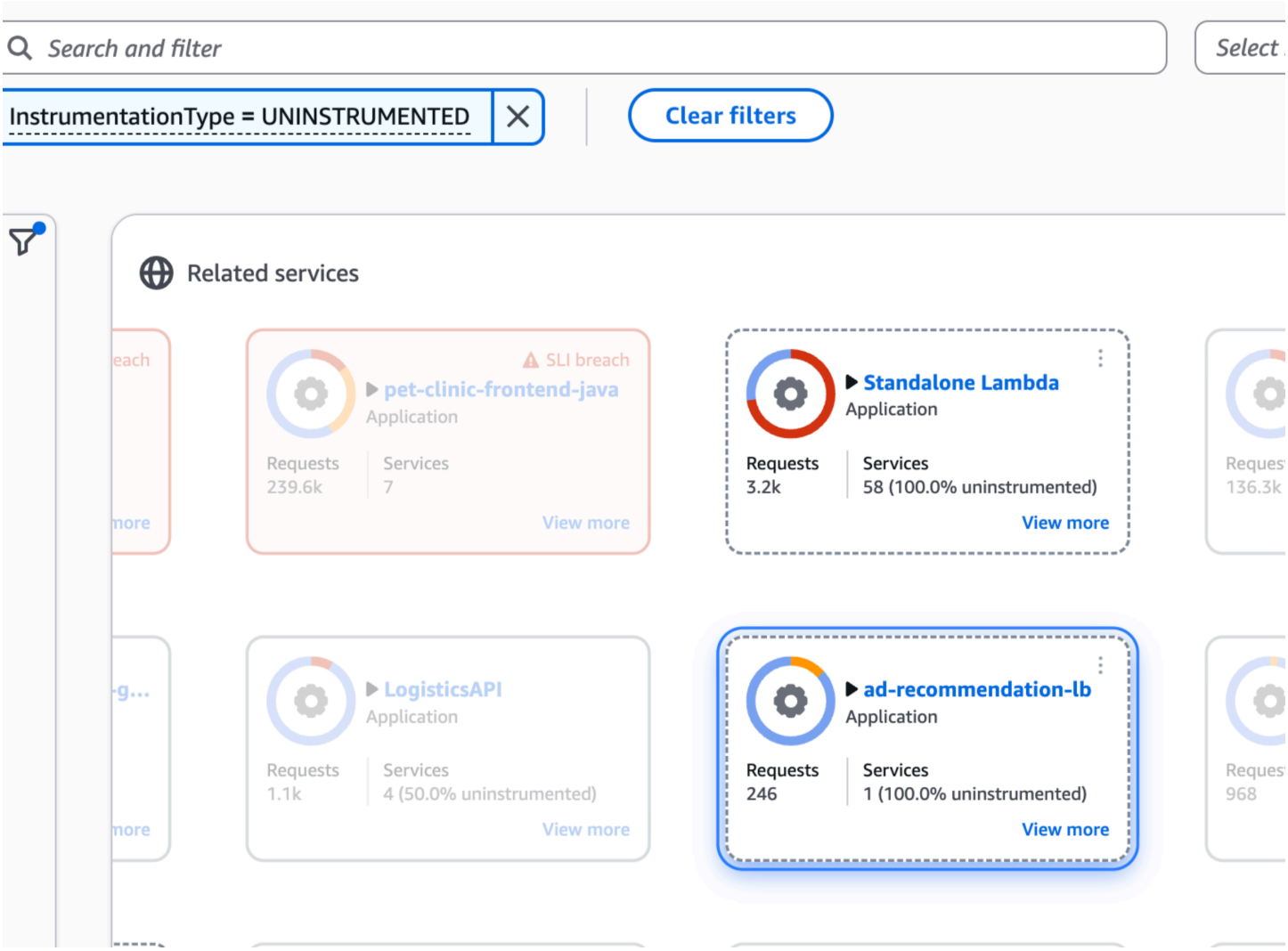
```
OTEL_RESOURCE_ATTRIBUTES="Application=PetClinic,Owner=Test,aws.application_signals.metric_resou
```

Viewing services within groups

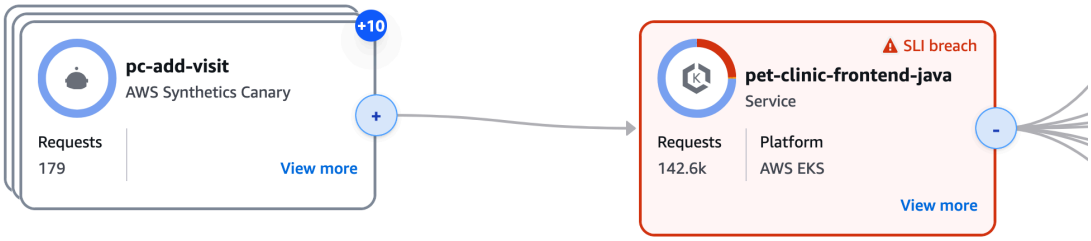
To view services and their dependencies in a group, click on the Group name. It will show a map of services inside the group. Each service node will show SLI health, metrics and platform details. Services with SLI breach are highlighted to be easily recognizable.



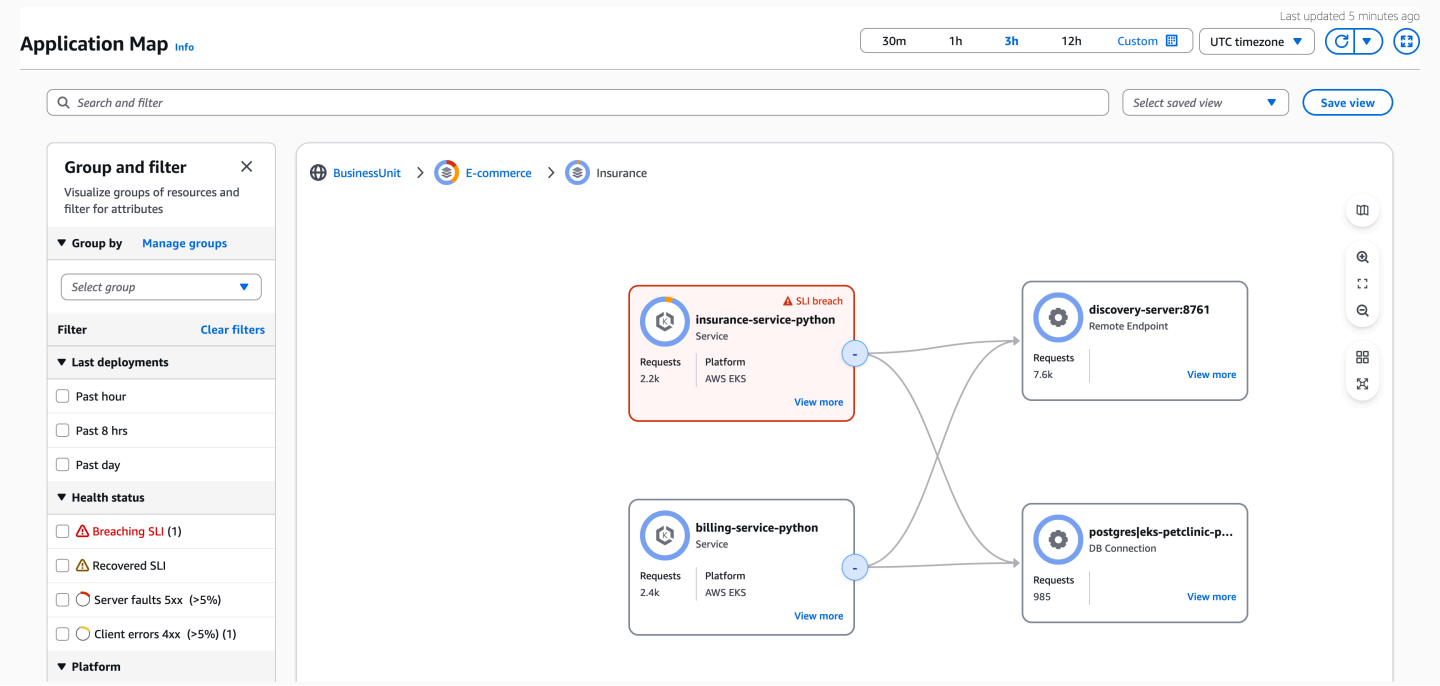
Un-instrumented services are displayed with a distinctive visual indicator (such as a dashed border or different color) to differentiate them from instrumented services. Hover over an un-instrumented service node to see instrumentation guidance and links to setup documentation.



All Canaries, RUM Clients and AWS Service nodes will be collapsed by default. If services in this group call services which are not part of this group, they will also be collapsed by default.

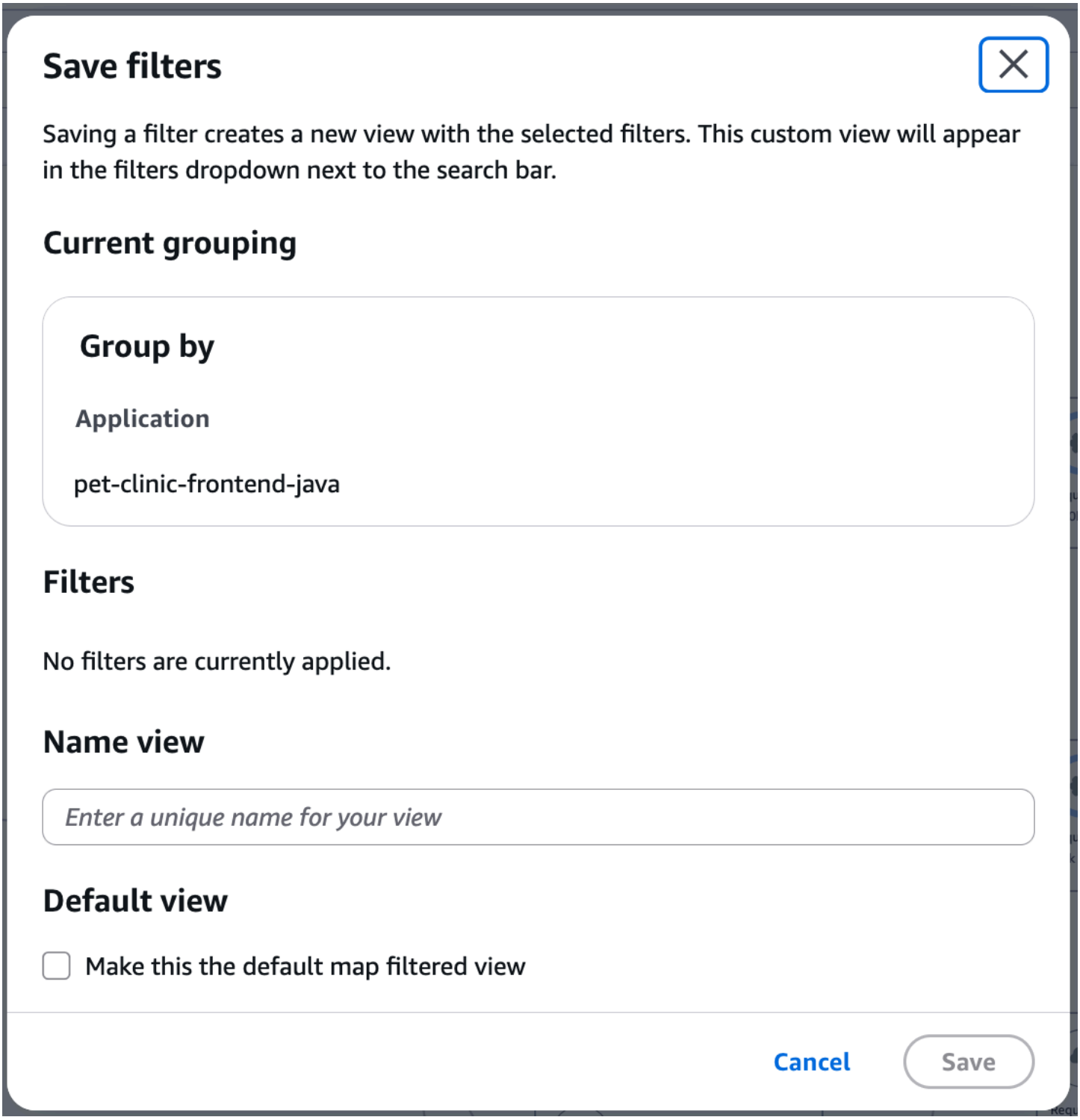


If your map is still too large to investigate effectively, you can apply nested grouping to narrow down your investigation. For example, after grouping services by **Business Unit**, if you still have too many services in a group, use the Group by dropdown to select **Team**, creating a nested grouping structure.

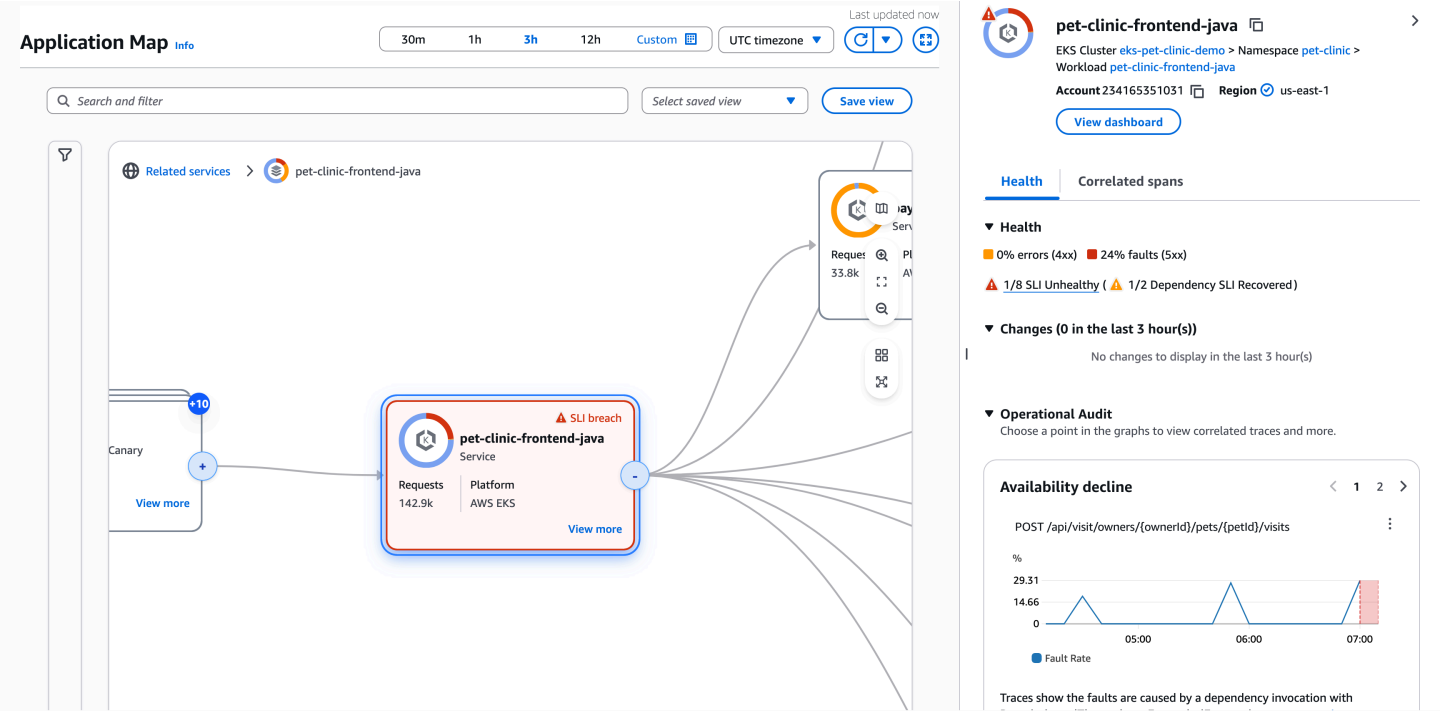


Service insights and details

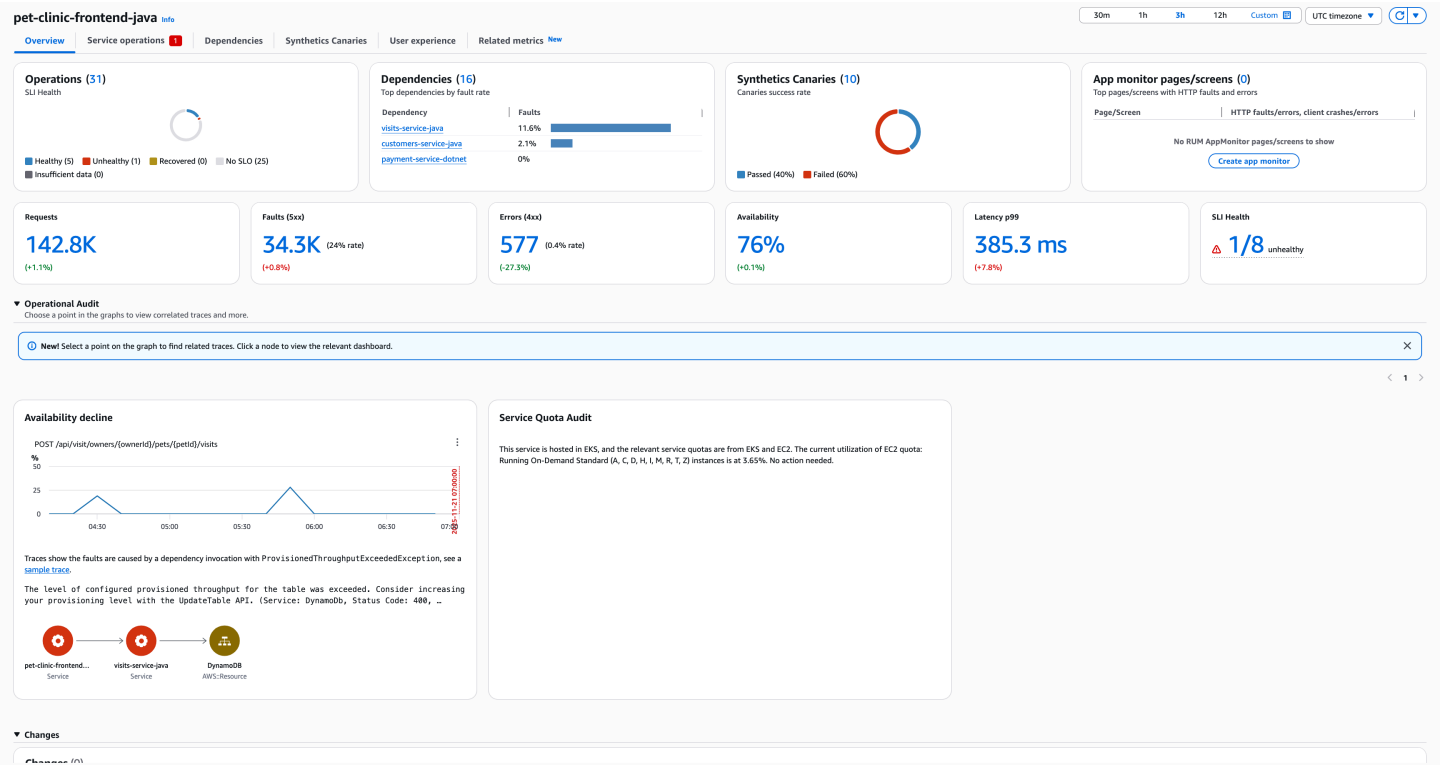
While on this page you can also click **Save view** next to search bar to save your view so next time you don't have to apply the same grouping and filtering again.



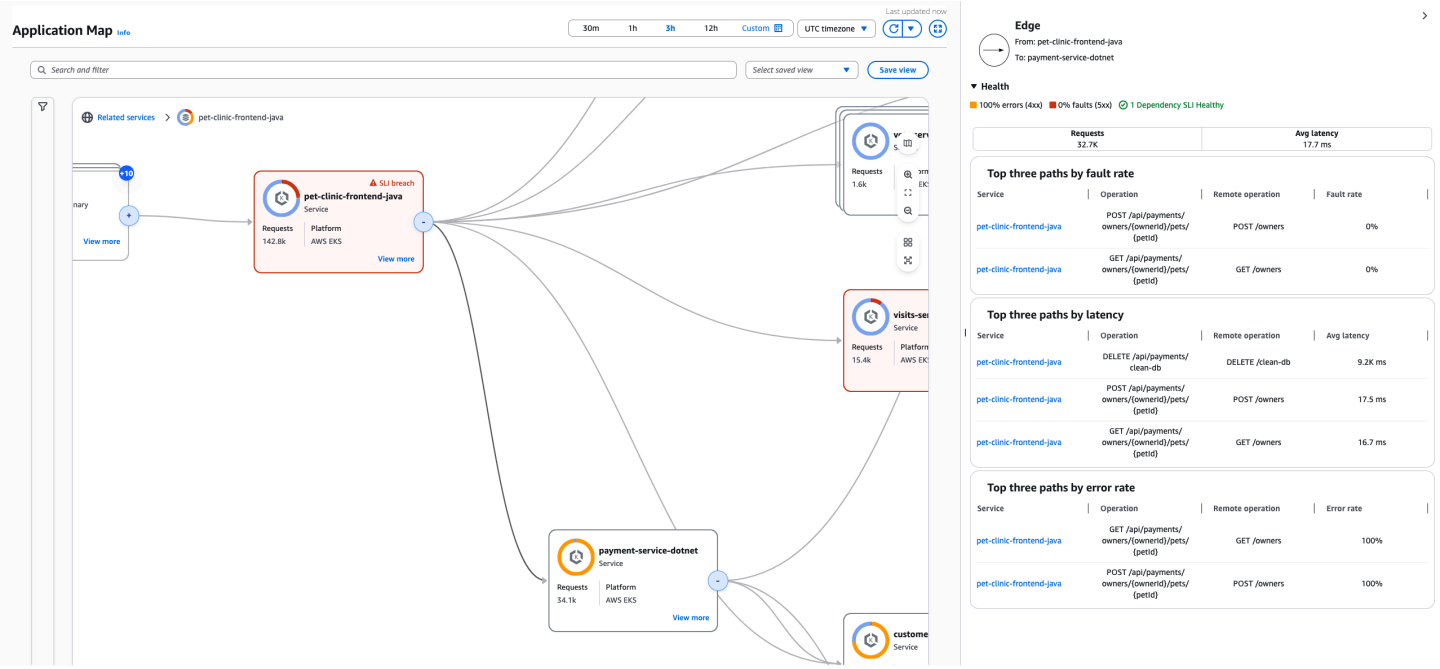
Click on **View more** in service node to view Service Audit, Change events, SLI health and Metrics graphs.



If you want to view service operation and other service detail, click on **View dashboard** to go to service overview page.



Alternatively you can click on Edge to view metrics of a specific dependency call of a service.



Change Events

Track change events across your application with Application Signals' automatic processing of CloudTrail events. Monitor configuration and deployment events for services and their dependencies, providing immediate context for operational analysis and troubleshooting. Change event detection is enabled alongside service discovery enablement through the CloudWatch Console or StartDiscovery API. For EKS services, deployment detection requires that the EKS services are instrumented with the Application Signals instrumentation SDK. Application Signals automatically correlates deployment times with performance changes, helping you quickly identify if recent deployments contributed to service issues. View change event history and impact across your services without additional configuration or setup requirements.

Audit findings

Discover critical insights through Application Signals' audit findings. The service analyzes your applications to report significant observations and potential problems, simplifying root cause analysis. These automated findings consolidate relevant traces, eliminating the need to navigate through multiple clicks. The audit system helps teams quickly identify issues and their underlying causes, enabling faster problem resolution.

For services running on Amazon Bedrock, Application Signals automatically monitors generative AI token usage patterns. The audit system detects anomalies in input and output token consumption, comparing current usage against historical baselines. When token usage exceeds normal patterns,

audit findings provide detailed analysis including token consumption trends, cost implications, and recommendations for optimization. This helps teams identify inefficient prompts, unexpected token spikes, and opportunities to reduce generative AI operational costs.

Cross-Account Observability on Application Map

Application Signals supports cross-account observability, allowing you to monitor and visualize services distributed across multiple AWS accounts in a single unified application map. This capability is essential for organizations with multi-account architectures following AWS best practices.

Key Capabilities:

- *Unified View*: View services from multiple AWS accounts in a single application map, providing a complete picture of your distributed application architecture.
- *Account Identification*: Each service node clearly displays its account ID and region, making it easy to identify service ownership and location.
- *Centralized Monitoring*: Monitor the health, performance, and SLO status of services across all connected accounts from a single monitoring account.
- *Cross-Account Filtering*: Filter and group services by account ID to focus on specific accounts or view cross-account interactions.

How It Works:

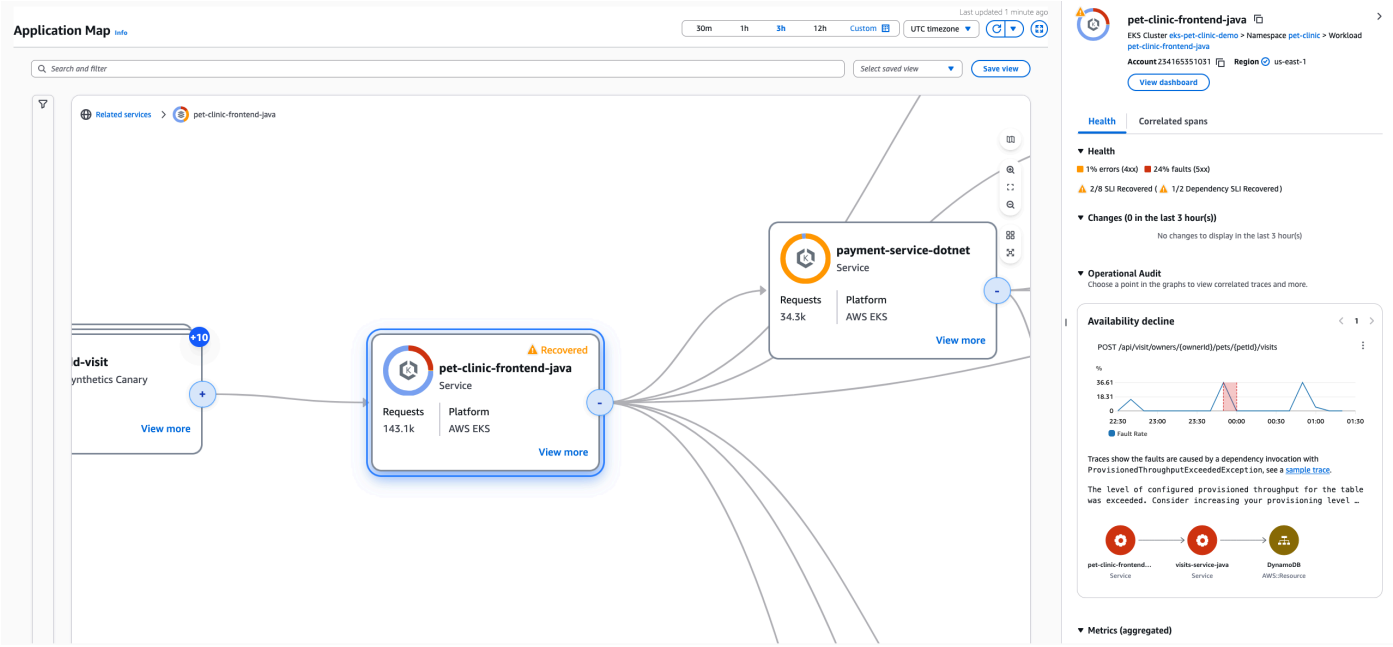
Application Signals uses AWS Organizations and cross-account sharing to enable observability across multiple accounts. To setup cross account observability please refer to CloudWatch cross-account observability.

View your application services

Service (Instrumented)

You can view your application services and the status of their SLOs and service level indicators (SLIs) in the **Application Map**. If you didn't create SLOs for a service, choose the **Create SLO** button below the service node.

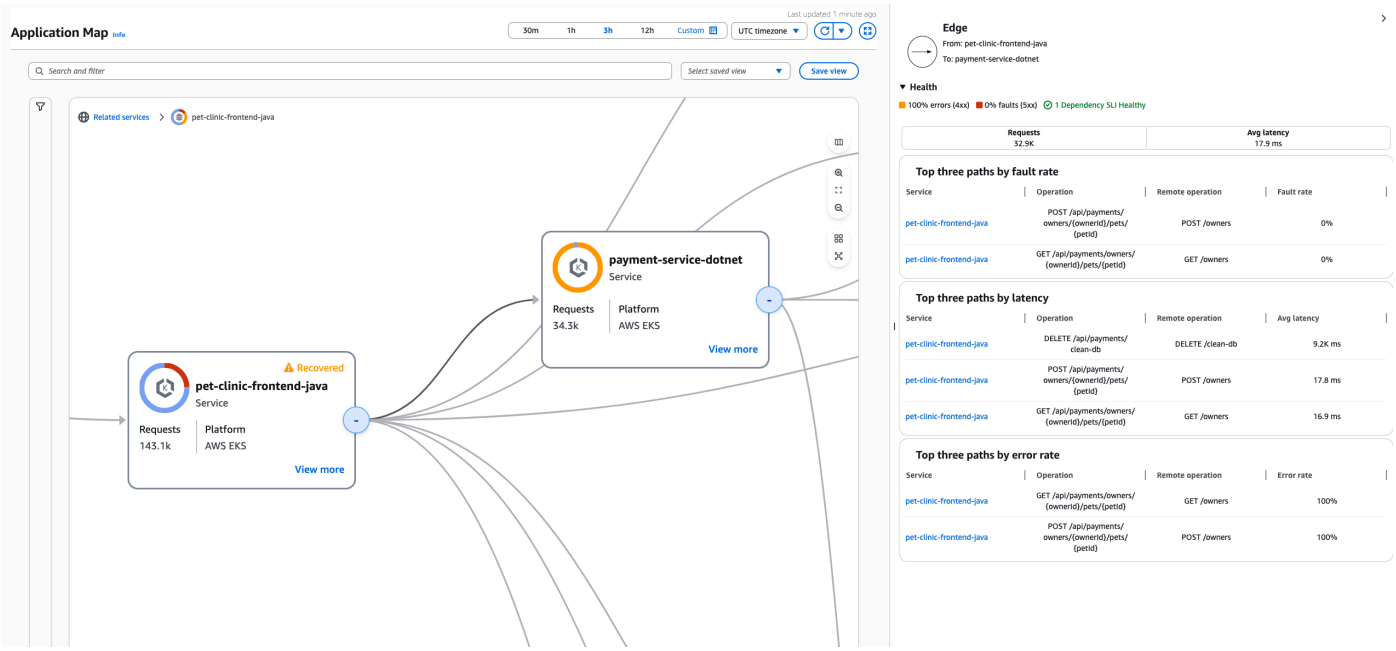
The **Application Map** displays all of your services. It also shows the customers and canaries that consume the service and the dependencies that your services calls, as shown in the following image:



When you select a service node, a pane opens displaying detailed service information:

- Total error and fault rate.
- The number of SLIs and SLOs that are `healthy` or `unhealthy`.
- The option to view more information about an SLO.
- The `Cluster`, `Namespace`, and `Workload` for services hosted in Amazon EKS, or Environment for services hosted in Amazon ECS or Amazon EC2. For Amazon EKS-hosted services, choose any link to open CloudWatch Container Insights.
- AccountId and region.
- The **Change** section showing recent change events and the last deployment time.
- The **Operational Audit** tab providing automated audit findings and recommendations.
- Service Metrics chart of Availability, latency, fault and errors.

Select an edge or connection between a service node and a downstream service or dependency node. This opens a pane containing top paths by fault rate, latency, and error rate, as shown in the following example image. Choose any link in the pane to open the Service details page and see detailed information for the chosen service or dependency.



When you select a edge node, a pane opens displaying detailed service information:

- Total request count, latency, error rate and fault rate
- Top path by fault rate
- Top path by latency
- Top path by error rate

Service (Un-instrumented)

Un-instrumented services appear on the Application Map even when they haven't been configured with Application Signals. These services are automatically discovered by leveraging Resource Explorer using application names and tags. The system can automatically detect up to 3,000 resources in your AWS account.

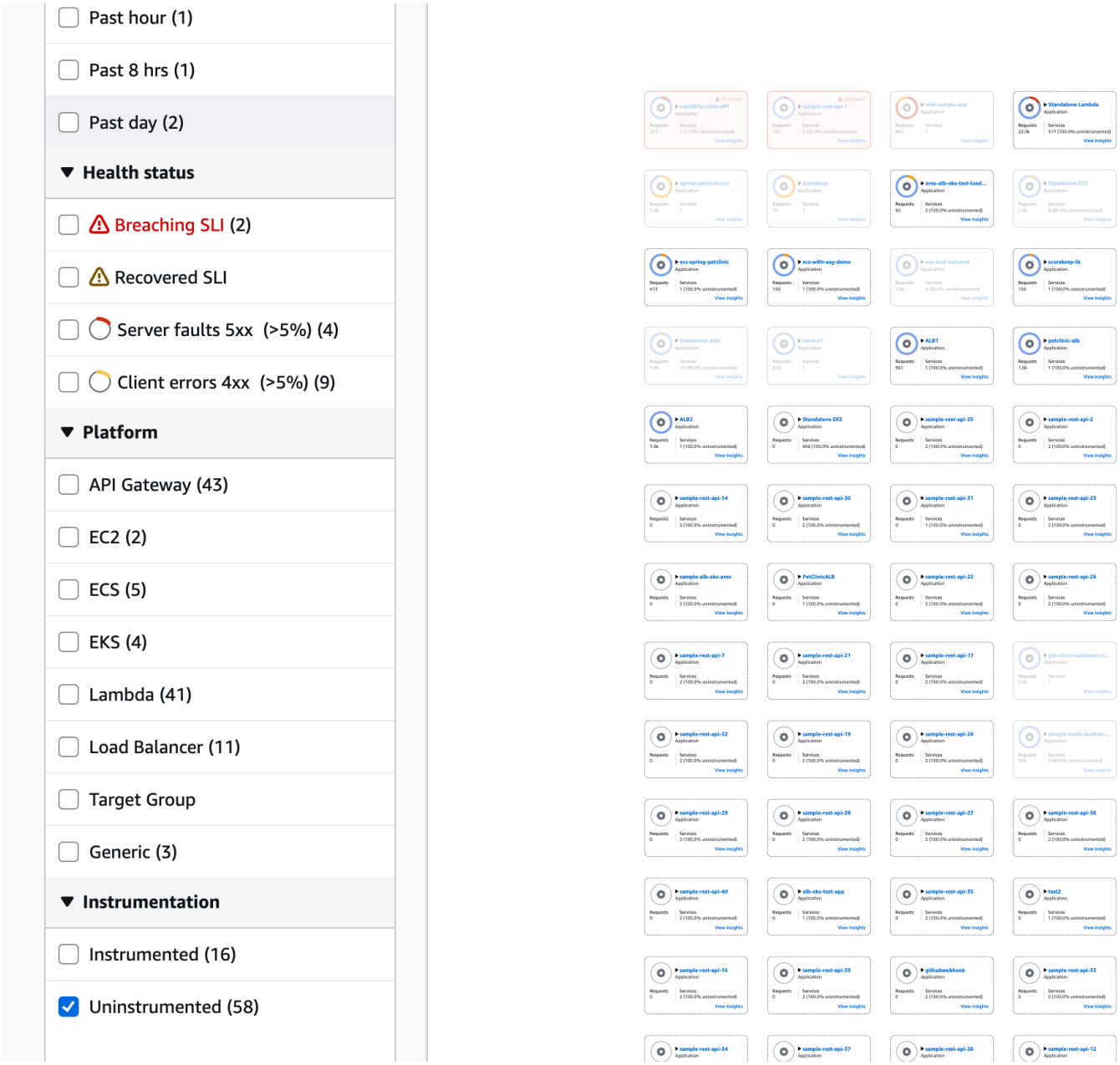
When you select an un-instrumented service node, a pane opens displaying:

- Service name and identification information
- AccountId and region where the service is detected
- Instrumentation status and guidance
- Call to action button "Enable Application Signals" that provides setup instructions
- Compute platform type (if detectable)

Un-instrumented services help you:

- Identify gaps in your observability coverage
- Prioritize which services to instrument next based on their position in your architecture
- Understand the complete application topology even before full instrumentation
- Plan instrumentation rollout across your organization

> **Note**
>
> Un-instrumented services display limited telemetry data since they don't actively send metrics or traces.

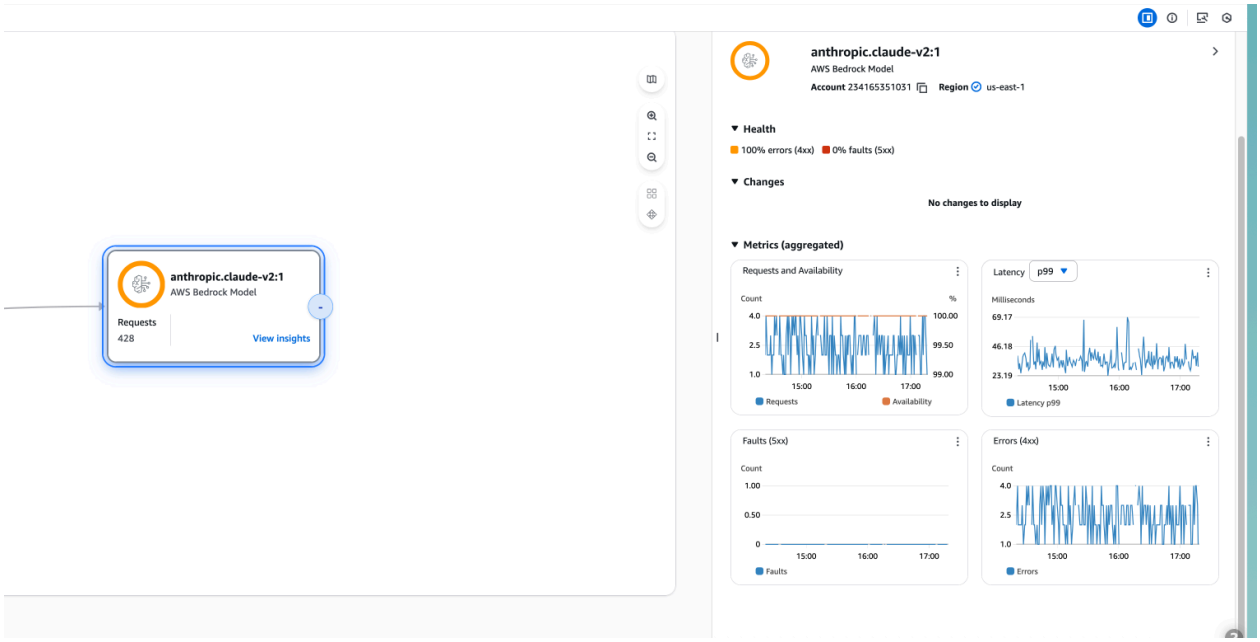


View dependencies

Your application dependencies are displayed on the application map, connected to the services that call them.

Choose a dependency node to open a pane containing error rate and fault rate, metrics chart for request, availability, latency, fault rate, and error rate.

If the dependency node is a service or resource, then the pane will display change events for the requested time range.

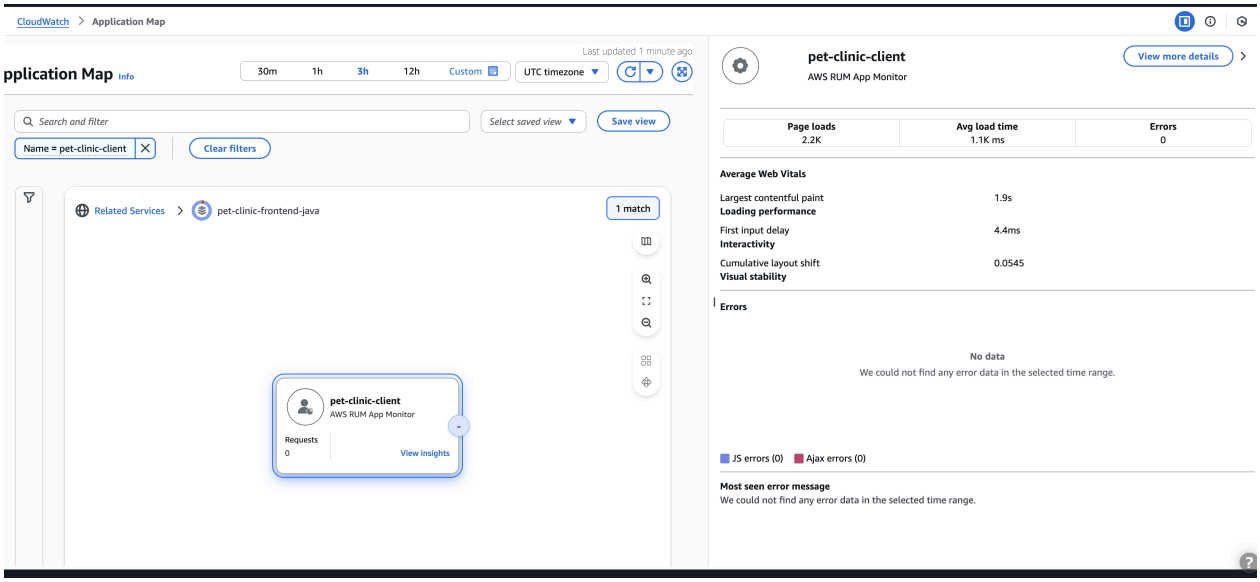


View clients

After you turn on X-Ray tracing for your CloudWatch RUM web clients, they display on the application map connected to services they call.

Choose a client node to open a pane displaying detailed client information:

- Metrics for page loads, average load time, errors, and average web vitals
- A graph displaying a breakdown of errors
- A link to display the client details in CloudWatch RUM



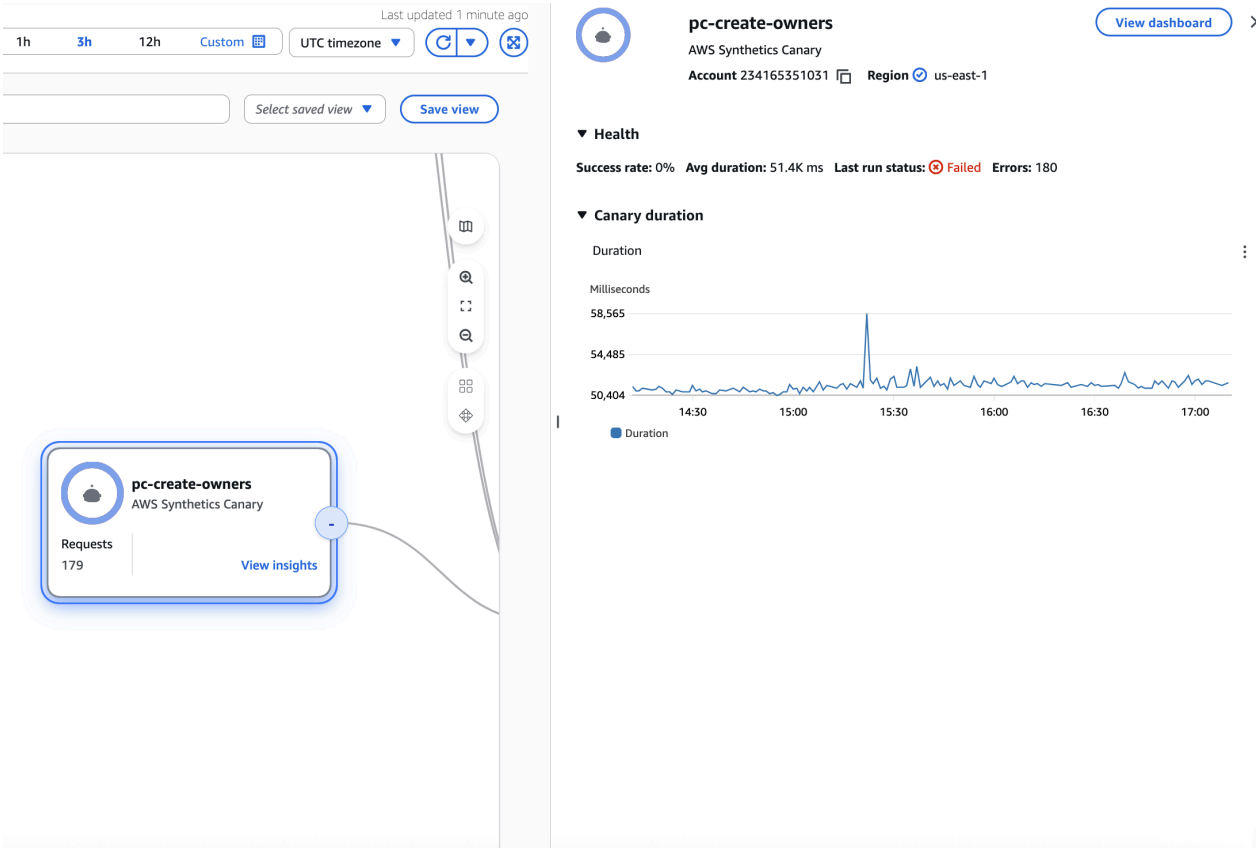
Choose **View dashboard** to open the canary details.

View synthetics canaries

To view canaries on your application map, turn on turn on X-Ray tracing for your CloudWatch Synthetics canaries. Once enabled, canaries will appear connected to their called services on the application map.

The system groups canaries together by default into a single expandable icon. The detailed canary information pane displays metrics, traces, and status information.

Choose a canary node to open a pane displaying detailed canary information, as shown in the following image:



Choose **View dashboard** to open the canary details.

Application observability for AWS Action

The Application Observability for AWS GitHub Action provides an end-to-end application observability investigation workflow that connects your source code and live production telemetry

data to AI agent. It leverages CloudWatch MCPs and generates custom prompts to provide the context that AI agents need for troubleshooting and applying code fixes.

The action sets up and configures CloudWatch Application Signals MCP Server and CloudWatch MCP Server, enabling them to access live telemetry data as troubleshooting context. You can use your preferred AI model - whether through your own API key, a third-party model, or Amazon Bedrock - for application performance investigations.

To get started, mention `@awsapm` in your GitHub issues to trigger the AI agent. The agent will troubleshoot production issues, implement fixes, and enhance observability coverage based on your live application data.

This action itself does not incur any direct costs. However, using this action may result in charges for AWS services and AI model usage. Please refer to the cost considerations documentation for detailed information about potential costs.

Getting Started

This action configures AI agents within your GitHub workflow by generating AWS-specific MCP configurations and custom observability prompts. You only need to provide IAM role to assume and a Bedrock Model ID you want to use, or API token from your existing LLM subscription. The example below demonstrates a workflow template that integrates this action with Anthropic's claude-code-base-action to run automated investigations.

Prerequisites

Before you begin, make sure you have the following:

- **GitHub Repository Permissions**: Write access or higher to the repository (required to trigger the action)
- **AWS IAM Role**: An IAM role configured with OpenID Connect (OIDC) for GitHub Actions with permissions for:
  - CloudWatch Application Signals and CloudWatch access
  - Amazon Bedrock model access (if using Bedrock models)
- **GitHub Token**: The workflow automatically uses GITHUB_TOKEN with the required permissions

Setup Steps

Step 1: Set up AWS Credentials

This action relies on the aws-actions/configure-aws-credentials action to set up AWS authentication in your GitHub Actions Environment. We recommend using OpenID Connect (OIDC) to authenticate with AWS. OIDC allows your GitHub Actions workflows to access AWS resources using short-lived AWS credentials so you do not have to store long-term credentials in your repository.

1. **Create an IAM Identity Provider**

   First, create an IAM Identity Provider that trusts GitHub's OIDC endpoint in the AWS Management Console:

   a. Open the IAM console

   b. Click **Identity providers** under **Access management**

   c. Click the **Add provider** button to add GitHub Identity provider if not yet created

   d. Select **OpenID Connect** type of Identity provider

   e. Enter `https://token.actions.githubusercontent.com` for the **Provider URL** input box

   f. Enter `sts.amazonaws.com` for the **Audience** input box

   g. Click the **Add provider** button

2. **Create an IAM Policy**

   Create an IAM policy with the required permissions for this action. See the Required Permissions section below for details.

3. **Create an IAM Role**

   Create an IAM role (for example, `AWS_IAM_ROLE_ARN`) in the AWS Management Console with the following trust policy template. This allows authorized GitHub repositories to assume the role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Federated": "arn:aws:iam::<AWS_ACCOUNT_ID>:oidc-provider/
token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "token.actions.githubusercontent.com:sub": "repo:<GITHUB_ORG>/
<GITHUB_REPOSITORY>:ref:refs/heads/<GITHUB_BRANCH>"
        }
      }
    }
  ]
}
```

Replace the following placeholders in the template:

- <AWS_ACCOUNT_ID> - Your AWS account ID
- <GITHUB_ORG> - Your GitHub organization name
- <GITHUB_REPOSITORY> - Your repository name
- <GITHUB_BRANCH> - Your branch name (for example, main)

4. **Attach the IAM Policy**

   In the role's Permissions tab, attach the IAM policy you created in step 2.

For more information about configuring OIDC with AWS, see the configure-aws-credentials OIDC Quick Start Guide.

Step 2: Configure Secrets and Add Workflow

1. **Configure Repository Secrets**

   Go to your repository → Settings → Secrets and variables → Actions.

   - Create a new repository secret named AWS_IAM_ROLE_ARN and set its value to the ARN of the IAM role you created in Step 1.
   - (Optional) Create a repository variable named AWS_REGION to specify your AWS region (defaults to us-east-1 if not set)

2. **Add the Workflow File**

The following is an example workflow that demonstrates using this action with Amazon Bedrock models. Create Application Observability Investigation workflow from this template in your GitHub Repository directory `.github/workflows`.

```
name: Application observability for AWS

on:
  issue_comment:
    types: [created, edited]
  issues:
    types: [opened, assigned, edited]

jobs:
  awsapm-investigation:
    if: |
      (github.event_name == 'issue_comment' && contains(github.event.comment.body,
 '@awsapm')) ||
      (github.event_name == 'issues' && (contains(github.event.issue.body, '@awsapm')
 || contains(github.event.issue.title, '@awsapm')))
    runs-on: ubuntu-latest

    permissions:
      contents: write        # To create branches for PRs
      pull-requests: write   # To post comments on PRs
      issues: write          # To post comments on issues
      id-token: write        # Required for AWS OIDC authentication

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_IAM_ROLE_ARN }}
          aws-region: ${{ vars.AWS_REGION || 'us-east-1' }}

      # Step 1: Prepare AWS MCP configuration and investigation prompt
      - name: Prepare Investigation Context
        id: prepare
        uses: aws-actions/application-observability-for-aws@v1
```

```
        with:
          bot_name: "@awsapm"
          cli_tool: "claude_code"

      # Step 2: Execute investigation with Claude Code
      - name: Run Claude Investigation
        id: claude
        uses: anthropics/claude-code-base-action@beta
        with:
          use_bedrock: "true"
          # Set to any Bedrock Model ID
          model: "us.anthropic.claude-sonnet-4-5-20250929-v1:0"
          prompt_file: ${{ steps.prepare.outputs.prompt_file }}
          mcp_config: ${{ steps.prepare.outputs.mcp_config_file }}
          allowed_tools: ${{ steps.prepare.outputs.allowed_tools }}

      # Step 3: Post results back to GitHub issue/PR (reuse the same action)
      - name: Post Investigation Results
        if: always()
        uses: aws-actions/application-observability-for-aws@v1
        with:
          cli_tool: "claude_code"
          comment_id: ${{ steps.prepare.outputs.awsapm_comment_id }}
          output_file: ${{ steps.claude.outputs.execution_file }}
          output_status: ${{ steps.claude.outputs.conclusion }}
```

Configuration Note:

- This workflow triggers automatically when `@awsapm` is mentioned in an issue or comment
- The workflow uses the `AWS_IAM_ROLE_ARN` secret configured in the previous step
- Update the model parameter in Step 2 to specify your preferred Amazon Bedrock model ID
- You can customize the bot name (for example, `@awsapm-prod`, `@awsapm-staging`) in the bot_name parameter to support different environments

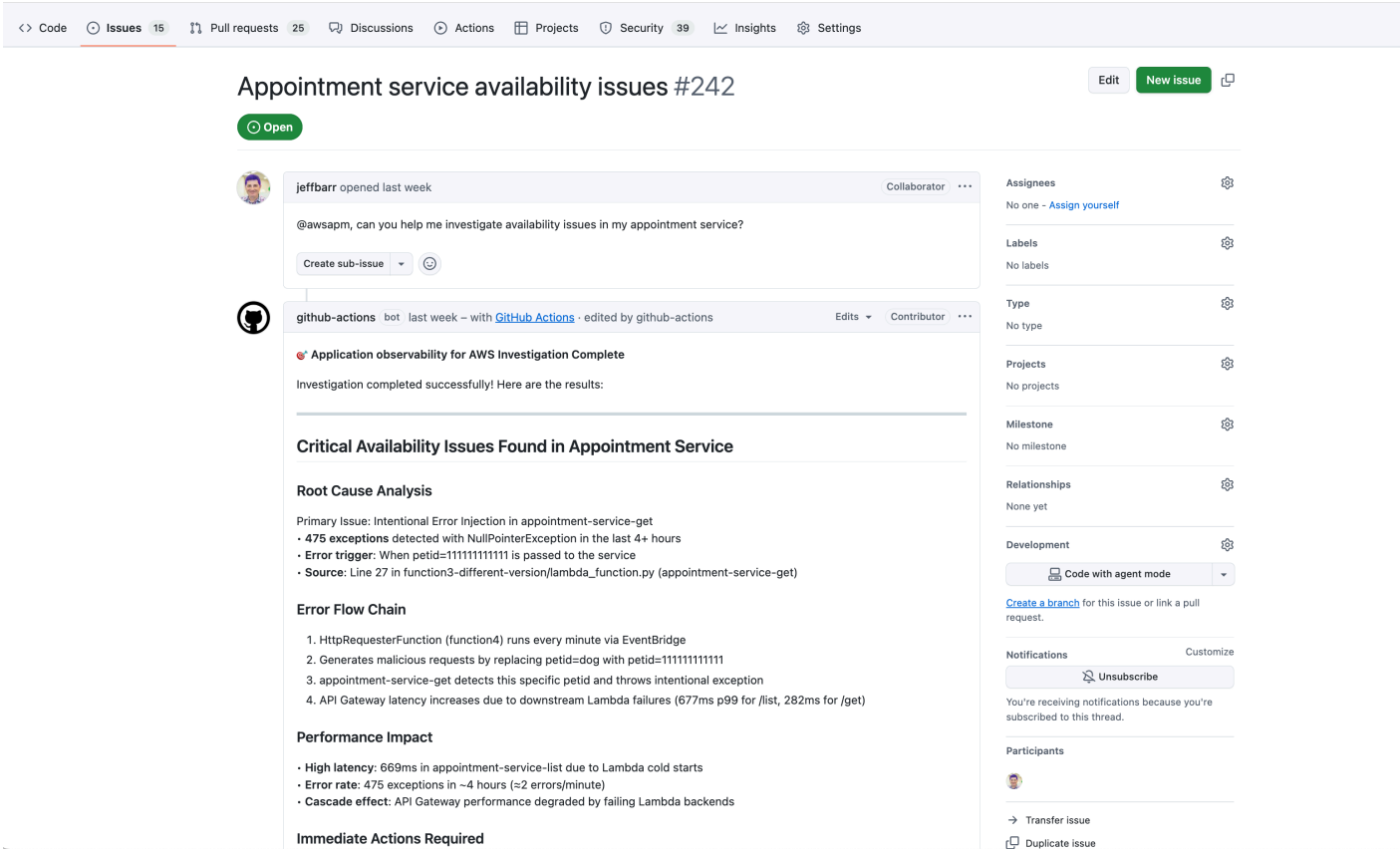
Step 3: Start Using the Action

After the workflow is configured, mention `@awsapm` in any GitHub issue to trigger an AI-powered investigation. The action will analyze your request, access live telemetry data, and provide recommendations or implement fixes automatically.

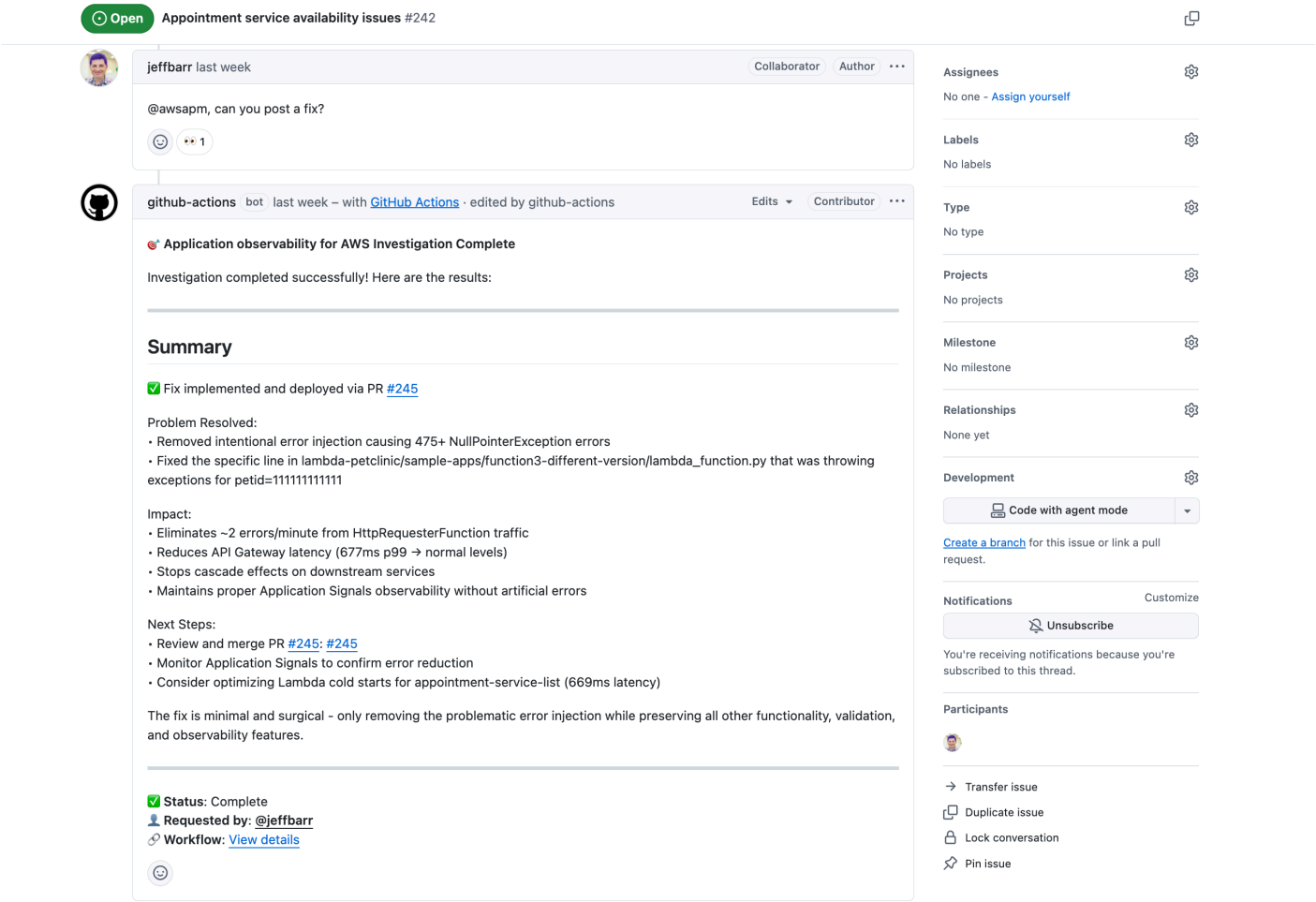
Example Use Cases:

1. Investigate performance issues and post and fix:

```
@awsapm, can you help me investigate availability issues in my
appointment service?
```




```
@awsapm, can you post a fix?
```



2. Enable instrumentation:

   ```
   @awsapm, please enable Application Signals for lambda-audit-service and create a PR with the required changes.
   ```

3. Query telemetry data:

   ```
   @awsapm, how many GenAI tokens have been consumed by my services in the past 24 hours?
   ```

What Happens Next:

1. The workflow detects the `@awsapm` mention and triggers the investigation
2. The AI agent accesses your live AWS telemetry data through the configured MCP servers
3. The agent analyzes the issue and either:
   - Posts findings and recommendations directly in the issue

- Creates a pull request with code changes (for instrumentation or fixes)

4. You can review the results and continue the conversation by mentioning @awsapm again with follow-up questions

Security

This action prioritizes security with strict access controls, OIDC-based AWS authentication, and built-in protections against prompt injection attacks. Only users with write access or higher can trigger the action, and all operations are scoped to the specific repository.

For detailed security information, including:

- Access control and permission requirements
- AWS IAM permissions and OIDC configuration
- Prompt injection risks and mitigations
- Security best practices

See the Security Documentation.

Configuration

Required Permissions

The IAM role assumed by GitHub Actions must have the following permissions.

Note: `bedrock:InvokeModel` and `bedrock:InvokeModelWithResponseStream` are only required if you're using Amazon Bedrock models

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "application-signals:ListServices",
                "application-signals:GetService",
                "application-signals:ListServiceOperations",
                "application-signals:ListServiceLevelObjectives",
                "application-signals:GetServiceLevelObjective",
```

```
                "application-signals:ListAuditFindings",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DescribeAlarmHistory",
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "logs:DescribeLogGroups",
                "logs:DescribeQueryDefinitions",
                "logs:ListLogAnomalyDetectors",
                "logs:ListAnomalies",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:GetQueryResults",
                "logs:FilterLogEvents",
                "xray:GetTraceSummaries",
                "xray:GetTraceSegmentDestination",
                "xray:BatchGetTraces",
                "xray:ListRetrievedTraces",
                "xray:StartTraceRetrieval",
                "servicequotas:GetServiceQuota",
                "synthetics:GetCanary",
                "synthetics:GetCanaryRuns",
                "s3:GetObject",
                "s3:ListBucket",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "bedrock:InvokeModel",
                "bedrock:InvokeModelWithResponseStream"
            ],
            "Resource": "*"
        }
    ]
}
```

Documentation

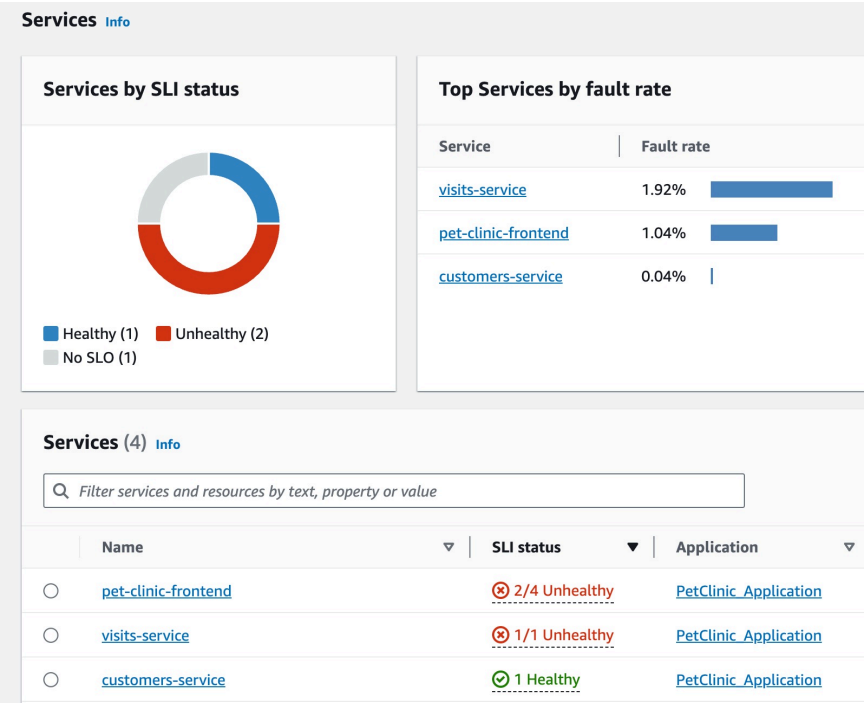
For more information, check out:

- CloudWatch Application Signals Documentation - Learn about CloudWatch Application Signals features and capabilities
- Application observability for AWS Action Public Documentation - Detailed guides and tutorials

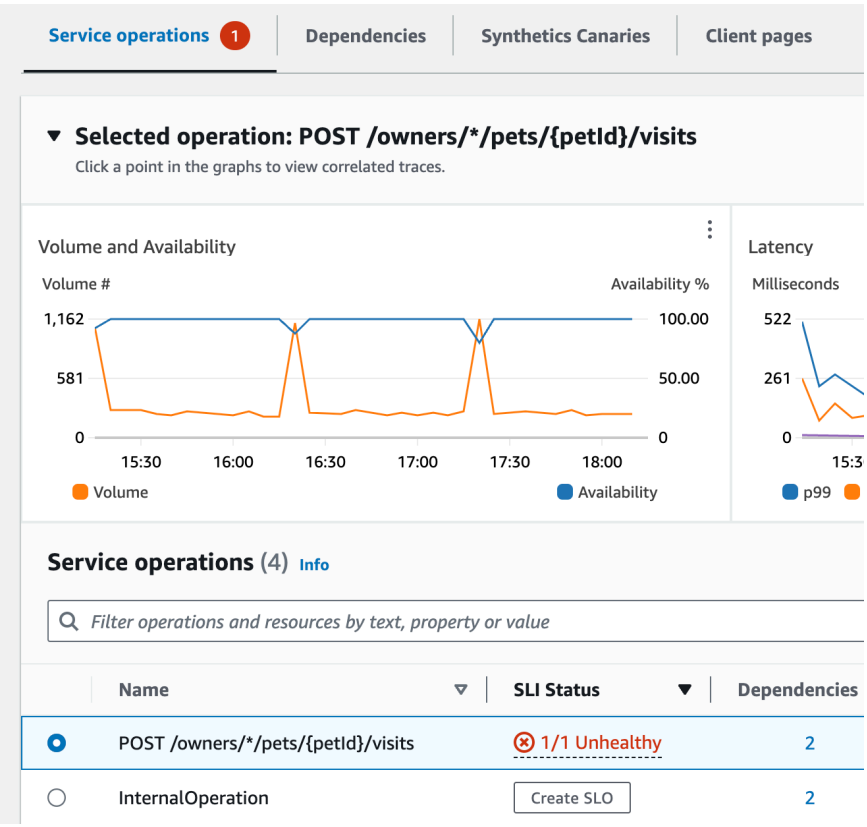
Example: Use Application Signals to resolve an operational health issue

The following scenario provides an example of how Application Signals can be used to monitor your services and identify service quality issues. Drill down to identify potential root causes and take action to resolve the issue. This example is focused on a pet clinic application composed of several microservices that call AWS services such as DynamoDB.

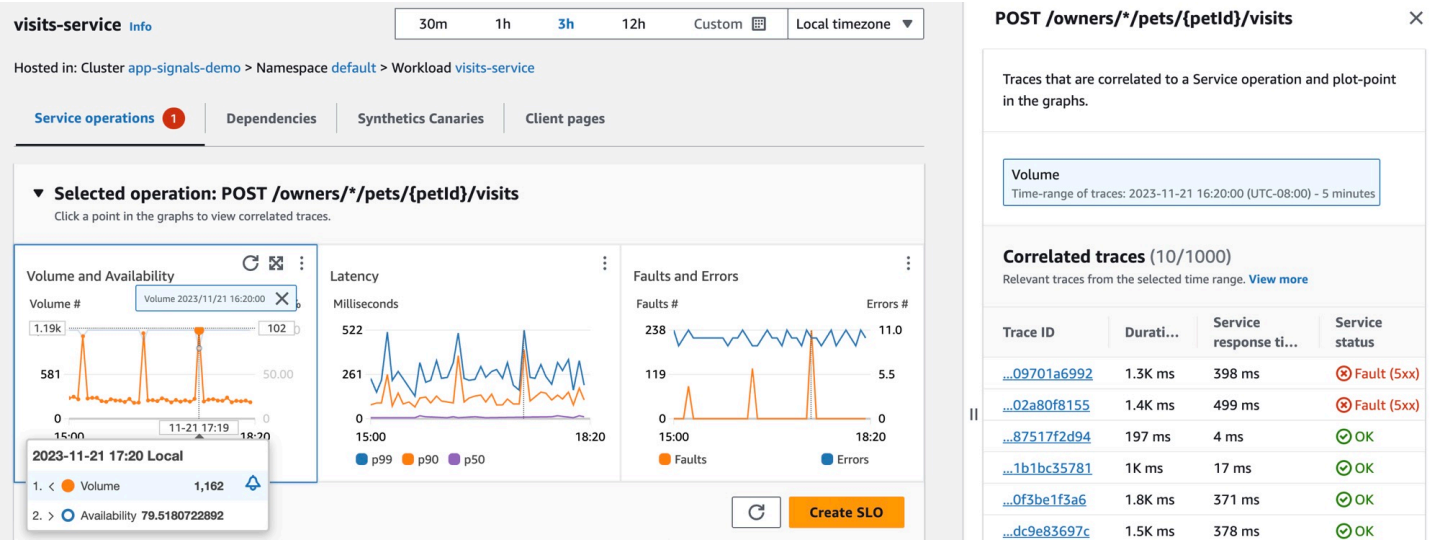
Jane is part of a DevOps team that oversees the operational health of a pet clinic application. Jane's team is committed to ensuring that the application is highly available and responsive. They use service level objectives (SLOs) to measure application performance against these business commitments. She receives an alert about several unhealthy service level indicators (SLIs). She opens the CloudWatch console and navigates to the Services page, where she sees several services in an unhealthy state.



At the top of the page, Jane sees that the `visits-service` is the top service by fault rate. She selects the link in the graph, which opens the Service detail page for the service. She sees that there is an unhealthy operation in the Service operations table. She selects this operation and sees in the Volume and Availability graph that there are periodic call volume spikes that seem to correlate to dips in availability.

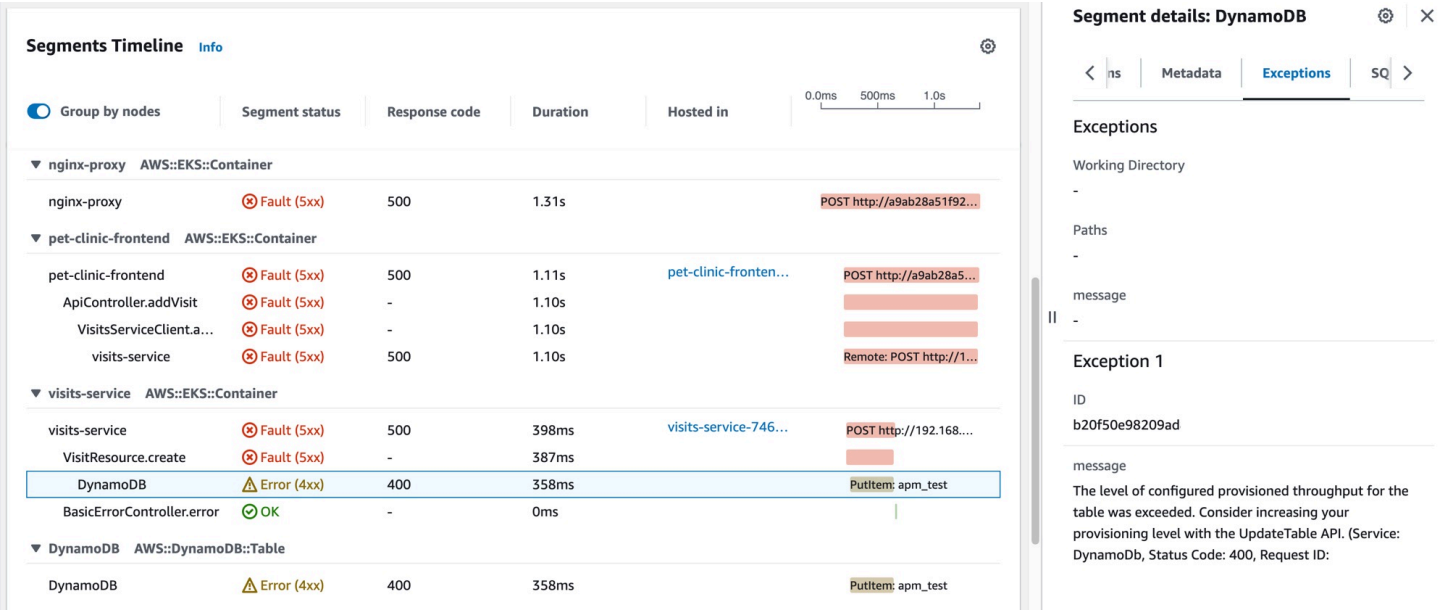


To look closer at the dips in service availability, Jane selects one of the availability data points in the graph. A drawer opens showing X-Ray traces that are correlated to the selected data point. She sees that there are multiple traces containing faults.



Jane selects one of the correlated traces with a fault status, which opens the X-Ray Trace detail page for the selected trace. Jane scrolls down to the Segments Timeline section and follows

the call path until she sees that calls to a DynamoDB table are returning errors. She selects the DynamoDB segment and navigates to the Exceptions tab of the right-side drawer.



Jane sees that a DynamoDB resource is misconfigured, resulting in errors during spikes in customer requests. The DynamoDB table's level of provisioned throughput is periodically exceeded, resulting in service availability issues and unhealthy SLIs. Based on this information, her team is able to configure a higher level of provisioned throughput and make sure high availability of the application.

Example: Use Application Signals to troubleshoot generative AI applications interacting with Amazon Bedrock models

You can use Application Signals to troubleshoot your generative AI applications that interact with Amazon Bedrock models. Application Signals streamlines this process by providing out-of-the-box telemetry data, offering deeper insights into your application's interactions with LLM models. It helps address key use cases such as:

- Model configuration issues
- Model usage costs
- Model latency
- Model response generation stopped reasons

Enabling Application Signals with LLM/GenAI Observability provides real-time visibility into your application's interactions with Amazon Bedrock services. Application Signals automatically generates and correlates performance metrics and traces for Amazon Bedrock API calls.

Application Signals currently support the following LLM Models from Amazon Bedrock.

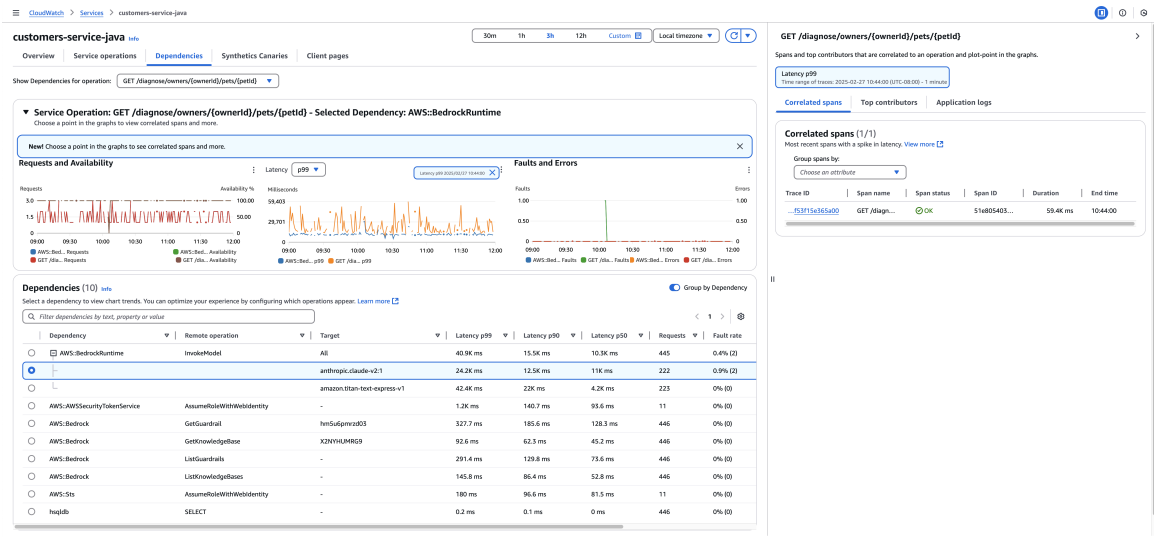
- AI21 Jamba
- Amazon Titan
- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI
- Nova

Fine-grained metrics and traces

For each Amazon Bedrock API call, Application Signals generates detailed performance metrics at the resource level, including:

- Model ID
- Guardrails ID
- Knowledge Base ID
- Bedrock Agent ID

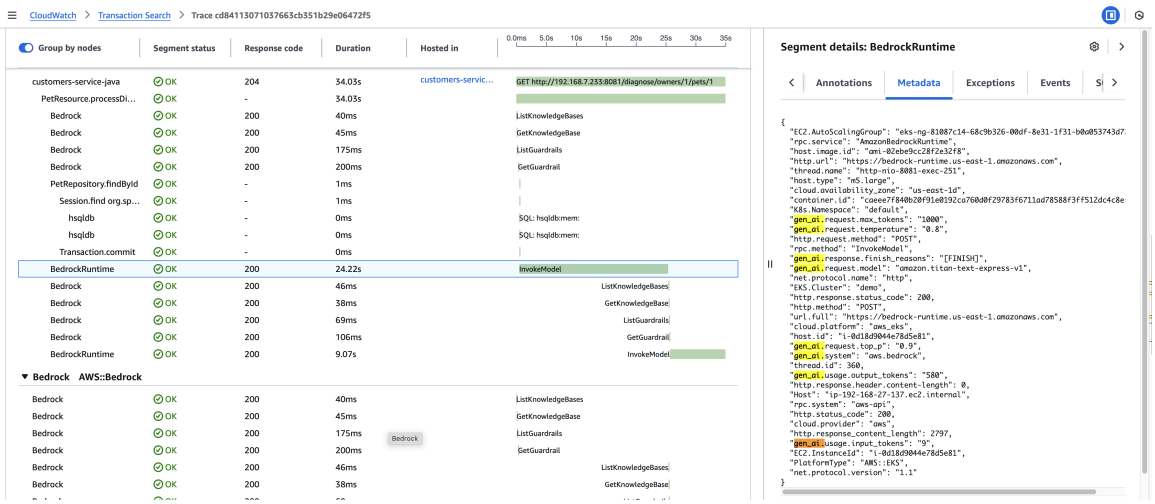
Additionally, correlated trace spans at the same level help provide a comprehensive view of request execution and dependencies.



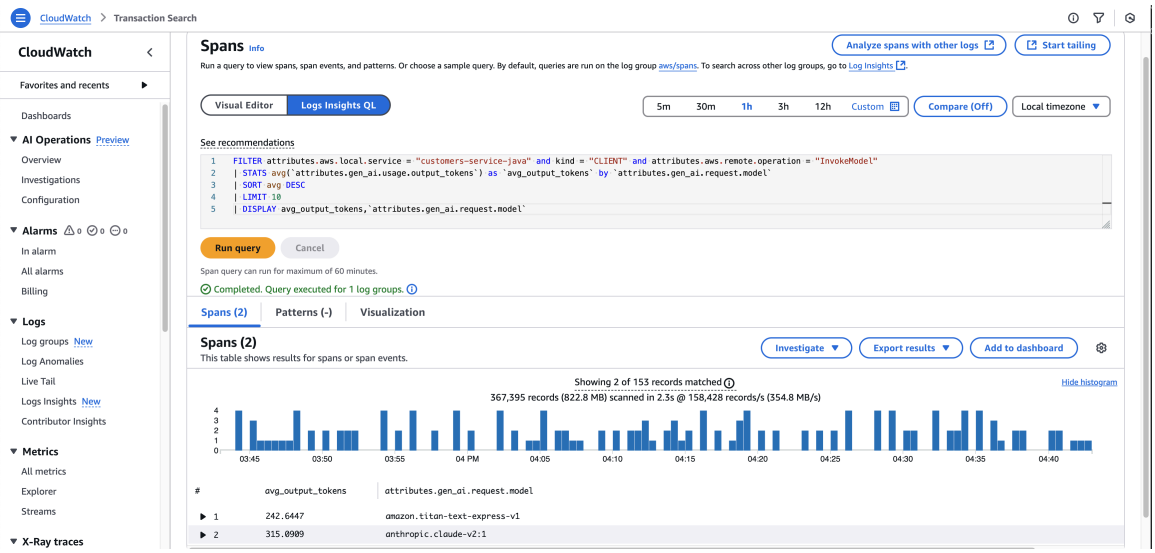
OpenTelemetry GenAI attributes support

Application Signals generates the following generative AI attributes for Amazon Bedrock API calls with OpenTelemetry semantic convention. These attributes help analyze model usage, cost, and response quality, and can be leveraged through Transaction Search for deeper insights.

- gen_ai.system
- gen_ai.request.model
- gen_ai.request.max_tokens
- gen_ai.request.temperature
- gen_ai.request.top_p
- gen_ai.usage.input_tokens
- gen_ai.usage.output_tokens
- gen_ai.response.finish_reasons



For example, you can use the analytic capability from Transaction Search to compare the token usage and cost across different LLM models for the same prompt, enabling cost-efficient model selection.



For more information, see Improve Amazon Bedrock Observability with CloudWatch Application Signals.

Metrics collected by Application Signals

Application Signals collects both standard application metrics and runtime metrics from the applications that you enable it for.

Standard application metrics relate to the most critical parts of service performance, latency and availability.

Runtime metrics track application metrics over time, including memory usage, CPU usage, and garbage collection. Application Signals displays the runtime metrics in the context of the services that you have enabled for Application Signals. When you have an operational issue, observing the runtime metrics can be useful to help you find the root cause of the issue. For example, you can see if spikes in latency in your service relate to spikes in a runtime metric.

Topics

- Standard application metrics collected
- Runtime metrics
- Disabling the collection of runtime metrics

Standard application metrics collected

Application Signals collects *standard application metrics* from the services that it discovers. These metrics relate to the most critical aspects of a service's performance: latency, faults, and errors. They can help you identify issues, monitor performance trends, and optimize resources to improve the overall user experience.

The following table lists the metrics collected by Application Signals. These metrics are sent to CloudWatch in the `ApplicationSignals` namespace.

Metric	Description
`Latency`	The delay before data transfer begins after the request is made. Units: Milliseconds
`Fault`	A count of both HTTP 5XX server-side faults and OpenTelemetry span status errors. Units: None
`Error`	A count of HTTP 4XX client-side errors. These are considered to be request errors that are not caused by service problems. Therefore, the `Availability` metric displayed on Application Signals dashboards does not regard these errors as service faults. Units: None

The `Availability` metric displayed on Application Signals dashboards is computed as **(1 - `Faults/Total`)*100**. Total responses includes all responses and is derived from `SampleCount(Latency)`. Successful responses are all responses without a 5XX error. 4XX responses are treated as successful when Application Signals calculates `Availability`.

Dimensions collected and dimension combinations

The following dimensions are defined for each of the standard application metrics. For more information about dimensions, see Dimensions.

Different dimensions are collected for *service metrics* and *dependency metrics*. Within the services discovered by Application Signals, when microservice A calls microservice B, microservice B is serving the request. In this case, microservice A emits dependency metrics and microservice B emits service metrics. When a client calls microservice A, microservice A is serving the request and emits service metrics.

Dimensions for service metrics

The following dimensions are collected for service metrics.

Dimension	Description
`Service`	The name of the service. The maximum value is 255 characters.
`Operation`	The name of the API operation or other activity. The maxiumum value is 1024 characters. You can set service level objectives on operations only if the operation name is 194 characters or fewer.
`Environment`	The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for `deployment.environment` in the `OTEL_ATTRIBUTE_RESOURCES` parameter. The maximum value is 259 characters.

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

- `[Environment, Service, Operation, [Latency, Error, Fault]]`
- `[Environment, Service, [Latency, Error, Fault]]`

Dimensions for dependency metrics

The following dimensions are collected for dependency metrics:

Dimension	Description
`Service`	The name of the service. The maximum value is 255 characters.
`Operation`	The name of the API operation or other operation. The maximum value is 1024 characters.
`RemoteService`	The name of the remote service being invoked. The maximum value is 255 characters.
`RemoteOperation`	The name of the API operation being invoked. The maximum value is 1024 characters.
`Environment`	The name of the environment where services are running. If services are not running on Amazon EKS, you can specify an optional custom value for `deployment.environment` in the `OTEL_ATTRIBUTE_RESOURCES` parameter. The maxiumum value is 259 characters.
`RemoteEnvironment`	The name of the environment where dependency services are running. The `RemoteEnvironment` parameter is automatically generated when a service calls a dependency and they are both running in the same cluster. Otherwise, `RemoteEnvironment` is neither generated

Dimension	Description
	nor reported in the service dependency's metrics. Currently only available on Amazon EKS and K8S platforms. The maximum value is 259 characters.
`RemoteResourceIdentifier`	The name of the resource invoked by a remote call. The `RemoteResourceIdentifier` parameter is automatically generated if service calls a remote AWS service. Otherwise, `RemoteResourceIdentifier` is neither generated nor reported in the service dependency's metrics. The maximum value is 1024 characters.
`RemoteResourceType`	The type of the resource that is invoked by a remote call. Required only if `RemoteResourceIdentifier` is defined. The maximum value is 1024 characters.

When you view these metrics in the CloudWatch console, you can view them using the following dimension combinations:

Running on Amazon EKS clusters

- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, Operation, RemoteService, RemoteOperation, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, [Latency, Error, Fault]]`

- `[Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, RemoteEnvironment, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, RemoteResourceIdentifier, RemoteResourceType, [Latency, Error, Fault]]`
- `[Environment, Service, RemoteService, RemoteOperation, [Latency, Error, Fault]]`
- `[RemoteService [Latency, Error, Fault]]`
- `[RemoteService, RemoteResourceIdentifier, RemoteResourceType [Latency, Error, Fault]]`

Runtime metrics

Application Signals uses the AWS Distro for OpenTelemetry SDK to automatically collect OpenTelemetry-compatible metrics from your Java and Python applications. To have runtime metrics collected, you must meet the following pre-requisites:

- Your CloudWatch agent must be version `1.300049.1` or later.
- If you use the Amazon CloudWatch Observability EKS add-on, it must be version `2.30-eksbuild.1` or later. If you update the add-on, you must restart your applications.
- For Java applications, you must be running `1.32.5` or later of the AWS Distro for OpenTelemetry SDK for Java.
- For Python applications, you must be running `0.7.0` or later of the AWS Distro for OpenTelemetry SDK for Python.
- For .Net applications, you must be running `1.6.0` or later of the AWS Distro for OpenTelemetry SDK for .Net.

Runtime metrics are not collected for Node.js applications.

Runtime metric are charged as part of Application Signals costs. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Note

Known issues
The runtime metrics collection in the Java SDK release v1.32.5 is known to not work with applications using JBoss Wildfly. This issue extends to the Amazon CloudWatch Observability EKS add-on, affecting versions `2.3.0-eksbuild.1` through `2.6.0-eksbuild.1`. The issue is fixed in Java SDK release `v1.32.6` and the Amazon CloudWatch Observability EKS add-on version `v3.0.0-eksbuild.1`.
If you are impacted, either upgrade the Java SDK version or disable your runtime metrics collection by adding the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` to your application.

Java runtime metrics

Application Signals collects the following JVM metrics from Java applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
`JVMGCDuration`	Aggregated metric for the duration of JVM garbage collection actions. Unit: Milliseconds	Sum, Average, Minimum, Maximum
`JVMGCOldGenDuration`	Aggregated metric for the duration of JVM garbage collection actions of the old generation. Available only in G1. Unit: Milliseconds	Sum, Average, Minimum, Maximum
`JVMGCYoungGenDuration`	Aggregated metric for the duration of JVM garbage collection actions of the young generation. Available only in G1. Unit: Milliseconds	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
JVMGCCount	Aggregated metric for the number of JVM garbage collection actions. Unit: None	Sum, Average, Minimum, Maximum
JVMGCOldGenCount	Aggregated metric for the number of JVM garbage collection actions of the old generation. Available only in G1. Unit: None	Sum, Average, Minimum, Maximum
JVMGCYoungGenCount	Aggregated metric for the number of JVM garbage collection actions of the young generation. Available only in G1. Unit: None	Sum, Average, Minimum, Maximum
JVMMemoryHeapUsed	The amount of memory heap used. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryUsedAfterLastGC	Amount of memory used, as measured after the most recent garbage collection event on this pool. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryOldGenUsed	The amount of memory used by the old generation. Unit: Bytes	Average, Minimum, Maximum
JVMMemorySurvivorSpaceUsed	The amount of memory heap used by the survivor space. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
JVMMemoryEdenSpaceUsed	The amount of memory used by the eden space. Unit: Bytes	Average, Minimum, Maximum
JVMMemoryNonHeapUsed	The amount of non-heap memory used. Unit: Bytes	Average, Minimum, Maximum
JVMThreadCount	The number of executing threads, including both daemon and non-daemon threads. Unit: None	Sum, Average, Minimum, Maximum
JVMClassLoaded	The number of classes loaded. Unit: None	Sum, Average, Minimum, Maximum
JVMCpuTime	The CPU time used by the process, as reported by the JVM. Unit: None (Nanoseconds)	Sum, Average, Minimum, Maximum
JVMCpuRecentUtilization	The recent CPU utilized by the process, as reported by the JVM. Unit: None	Average, Minimum, Maximum

Python runtime metrics

Application Signals collects the following metrics from Python applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
PythonProcessGCCount	The total number of objects currently being tracked. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessGCGen0Count	The number of objects currently being tracked in Generation 0. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessGCGen1Count	The number of objects currently being tracked in Generation 1. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessGCGen2Count	The number of objects currently being tracked in Generation 2. Unit: None	Sum, Average, Minimum, Maximum
PythonProcessVMSMemoryUsed	The total amount of virtual memory used by the process. Unit: Bytes	Average, Minimum, Maximum
PythonProcessRSSMemoryUsed	The total amount of non-swapped physical memory used by the process. Unit: Bytes	Average, Minimum, Maximum
PythonProcessThreadCount	The number of threads currently used by the process. Unit: None	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
`PythonProcessCpuTime`	The CPU time used by the process. Unit: Seconds	Sum, Average, Minimum, Maximum
`PythonProcessCpuUtilization`	The CPU utilization of the process. Unit: None	Average, Minimum, Maximum

.Net runtime metrics

Application Signals collects the following metrics from .Net applications that you enable for Application Signals. All runtime metrics are sent to CloudWatch in the `ApplicationSignals` namespace, and are collected with the `Service` and `Environment` dimension set.

Metric name	Description	Meaningful statistics
`DotNetGCGen0Count`	The total number of garbage collection metrics tracked in Generation 0 since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen1Count`	The total number of garbage collection metrics tracked in Generation 1 since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen2Count`	The total number of garbage collection metrics tracked in Generation 2 since the process started. Unit: None	Sum, Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
`DotNetGCDuration`	The total amount of time paused in garbage collection since the process started. Unit: None	Sum, Average, Minimum, Maximum
`DotNetGCGen0HeapSize`	The heap size (including fragmentation) of Generation 0 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
`DotNetGCGen1HeapSize`	The heap size (including fragmentation) of Generation 1 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
DotNetGCGen2HeapSize	The heap size (including fragmentation) of Generation 2 observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
DotNetGCLOHHeapSize	The large object heap size (including fragmentation) observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum

Metric name	Description	Meaningful statistics
DotNetGCPOHHeapSize	The pinned object heap size (including fragmentation) observed during the latest garbage collection. **Note** This metric is only available after the first Garbage Collection is complete. Unit: Bytes	Average, Minimum, Maximum
DotNetThreadCount	The number of thread pool threads that currently exist. Unit: None	Average, Minimum, Maximum
DotNetThreadQueueLength	The number of work items that are currently queued to be processed by the thread pool. Unit: None	Average, Minimum, Maximum

Disabling the collection of runtime metrics

Runtime metrics are collected by default for Java and Python applications that are enabled for Application Signals. If you want to disable the collection of these metrics, follow the instructions in this section for your environment.

Amazon EKS

To disable runtime metrics in Amazon EKS applications at the application level, add the following environment variable to your workload specification.

```
env:
```

```
    - name: OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED
      value: "false"
```

To disable runtime metrics in Amazon EKS applications at the cluster level, apply the configuration to the advanced configuration of your Amazon CloudWatch Observability EKS add-on.

```
{
  "agent": {
    "config": {
      "traces": {
        "traces_collected": {
          "application_signals": {

          }
        }
      },
      "logs": {
        "metrics_collected": {
          "application_signals": {

          }
        }
      }
    },
    "manager": {
      "autoInstrumentationConfiguration": {
        "java": {
          "runtime_metrics": {
            "enabled": false
          }
        },
        "python": {
          "runtime_metrics": {
            "enabled": false
          }
        },
        "dotnet": {
          "runtime_metrics": {
            "enabled": false
          }
        }
      }
    }
```

```
  }
}
```

Amazon ECS

To disable runtime metrics in Amazon ECS applications, add the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` in the new task definition revision and redeploy the application.

EC2

To disable runtime metrics in Amazon EC2 applications, add the environment variable `OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED=false` before the application starts.

Kubernetes

To disable runtime metrics in Kubernetes applications at the application level, add the following environment variable to your workload specification.

```
env:
    - name: OTEL_AWS_APPLICATION_SIGNALS_RUNTIME_ENABLED
      value: "false"
```

To disable runtime metrics in Kubernetes applications at the cluster level, use the following:

```
helm upgrade ... \
--set-string
 manager.autoInstrumentationConfiguration.java.runtime_metrics.enabled=false \
--set-string
 manager.autoInstrumentationConfiguration.python.runtime_metrics.enabled=false \
-\-set-string
 manager.autoInstrumentationConfiguration.dotnet.runtime_metrics.enabled=false
```

Custom metrics with Application Signals

To monitor application performance and availability, Application Signals collects standard metrics (faults, errors, and latency) and runtime metrics from discovered applications after you enable it.

Custom Metrics add valuable context to your application monitoring and help expedite troubleshooting. You can use them to:

- Customize analysis of telemetry data

- Identify root causes of issues
- Make precise business and operational decisions quickly

Application Signals lets you view and correlate custom metrics generated from a service with standard and runtime metrics. For example, an application could emit metrics for request size and cache miss count. These custom metrics provide more granular insight into performance issues, helping you diagnose and resolve availability drops and latency spikes faster.

Topics

- Configuring custom metrics to Application Signals
- Viewing custom metrics in Application Signals
- Frequently asked questions (FAQs)

Configuring custom metrics to Application Signals

You can generate custom metrics from your application using two methods: *OpenTelemetry metrics* and *Span metrics*.

Topics

- OpenTelemetry metrics
- Span metrics

OpenTelemetry metrics

To use custom OpenTelemetry metrics with Application Signals, you must use either the CloudWatch Agent or OpenTelemetry Collector. Custom OpenTelemetry metrics allow you to create and export metrics directly from your application code using the OpenTelemetry Metrics SDK.

1. Onboard service to Application Signals.
2. Configure the agent or collector.
   - When using the CloudWatch agent, you must configure `metrics_collected` with an `otlp`. For example, `cloudwatch-config.json`

```
{
```

```
  "traces": {
    "traces_collected": {
      "application_signals": {}
    }
  },
  "logs": {
    "metrics_collected": {
      "application_signals": {},
      "otlp": {
        "grpc_endpoint": "0.0.0.0:4317",
        "http_endpoint": "0.0.0.0:4318"
      }
    }
  }
}
```

- When using OpenTelemetry Collector, configure a metrics pipeline. You must use CloudWatch EMF Exporter for OpenTelemetry Collector and enable Resource Attributes to Metric Labels. It's recommended to configure `dimension_rollup_option: NoDimensionRollup` to avoid emitting many metric aggregations. For example, `config.yaml`:

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  awsemf:
    region: $REGION
    namespace: $NAMESPACE
    log_group_name:$LOG_GROUP_NAME
    resource_to_telemetry_conversion:
      enabled: true
    dimension_rollup_option: "NoDimensionRollup"

  otlphttp/traces:
    compression: gzip
    traces_endpoint: https://xray.$REGION.amazonaws.com/v1/traces
    auth:
```

```
      authenticator: sigv4auth/traces

extensions:
  sigv4auth/logs:
    region: "$REGION"
    service: "logs"
  sigv4auth/traces:
    region: "$REGION"
    service: "xray"

processors:
  batch:

service:
  telemetry:
  extensions: [sigv4auth/logs, sigv4auth/traces]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [awsemf]
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp/traces]
```

3. Configure the environment. When there are multiple services with the same service name and to accurately correlate Application Signals metrics to the correct service name, it's recommended to configure the resource attribute `deployment.environment.name`. Configuring this resource attribute is commonly done through the environment variables.

```
OTEL_RESOURCE_ATTRIBUTES="service.name=$YOUR_SVC_NAME,deployment.environment.name=
$YOUR_ENV_NAME"
```

4. Configure metric export to the CloudWatch agent or OpenTelemetry Collector. You can use one of the following approach:

   - (Recommended) Custom export pipeline – In the application code, create a dedicated MeterProvider exporting to the configured agent or collector endpoint. For example:

```
Resource resource = Resource.getDefault().toBuilder()
        .put(AttributeKey.stringKey("service.name"), serviceName)
```

```
        .put(AttributeKey.stringKey("deployment.environment.name"), environment)
        .build();

MetricExporter metricExporter = OtlpHttpMetricExporter.builder()
        .setEndpoint("http://localhost:4318/v1/metrics")
        .build();

MetricReader metricReader = PeriodicMetricReader.builder(metricExporter)
        .setInterval(Duration.ofSeconds(10))
        .build()

SdkMeterProvider meterProvider = SdkMeterProvider.builder()
    .setResource(resource)
    .registerMetricReader()
    .build();

Meter meter = meterProvider.get("myMeter");
```

- Agent-based export – Configure the agent environment variables OTEL_METRICS_EXPORTER and OTEL_EXPORTER_OTLP_METRICS_ENDPOINT. For example:

  ```
  OTEL_METRICS_EXPORTER=otlp
  OTEL_EXPORTER_OTLP_METRICS_ENDPOINT=http://localhost:4318/v1/metrics
  ```

  In application code, rely on global MeterProvider created by the agent. For example:

  ```
  Meter meter = GlobalOpenTelemetry.getMeter("myMeter");
  ```

5. Using the OTEL Metrics SDK in the application code, add the OTEL Metrics. For example, to add the OTEL metrics in Python:

   ```
   counter = meter.counterBuilder("myCounter").build();
   counter.add(value);
   counter.add(value, Attributes.of(AttributeKey.stringKey("Operation"),
    "myOperation"));
   ```

   Adding the Operation attribute is not required, but can be useful for correlating Application Signals Service Operations to Custom OpenTelemetry Metrics.

Span metrics

Custom Span metrics currently only work with Transaction Search. With custom Span metrics, you can:

- Create metrics using Metrics Filters
- Process span attributes added in application code
- Use the OpenTelemetry Traces SDK for implementation

1. Enable Application Signals monitoring with Transaction Search. For more information, see Transaction Search.

   To make sure 100% metric sampling, it's recommended to send 100% of spans to the endpoint.
2. Add span attributes using the OTEL Traces SDK. There are two ways:

   - [Recommended] Add attributes to automatically generated spans. For example:

     ```
     Span.current().setAttribute("myattribute", value);
     ```

   - Add attributes to manually generated spans. For example:

     ```
     Span span = tracer.spanBuilder("myspan").startSpan();
     try (Scope scope = span.makeCurrent()) {
        span.setAttribute("myattribute",  value);
     }
     ```
3. Create a metric filter with the following values. For information on how to create a metric filter, see Create a metric filter for a log group.

   - Log Group – aws/spans
   - Filter pattern – { $.attributes.['myattribute'] = * }
   - Metric name – myattribute (The values must be an exact match or span correlation will not work
   - Metric value – $.attributes.['myattribute']
   - Dimensions – Field Name: Service, Field Value: $.attributes.['aws.local.service'], Field Name: Environment, Field Value: $.attributes.['aws.local.environment'], and Field Name: Operation, Field Value: $.attributes.['aws.local.operation']

> **Note**
>
> When you add attributes to manually generated spans, you cannot set `Operation` because `aws.local.operation` will not be present in span data.

Viewing custom metrics in Application Signals

You can now view custom metrics for services and operations in the Application Signals console:

- Select a service from the **Services** list to see the new **Related Metrics** tab
- View standard metrics, runtime metrics, and related metrics for your selected service
- Filter and select multiple metrics from the list
- Graph selected metrics to identify correlations and root causes of issues

For more information on Related Metrics, see View Related metrics.

Frequently asked questions (FAQs)

What is the impact of not adding the configuration for environment for custom metrics?

Application Signals configures the `deployment.environment.name` resource attribute to disambiguate applications. Application Signals cannot correlate custom metrics generated from two different services with the same name to the correct service without disambiguation.

To add environment configuration to your application, see OpenTelemetry metrics.

Are there any limits for metrics filters?

You can only create up to 100 metrics filters per CloudWatch Logs log group. Each metric defined can have up to 3 dimensions. You can view limits for metrics filters here OpenTelemetry metrics.

Why aren't metric graphs appearing in the metrics table?

The solution depends on your metric type:

- Custom metrics – See Configuring custom metrics to Application Signals to verify the metric configuration
- Standard or runtime metrics – See Troubleshooting your Application Signals installation

Service level objectives (SLOs)

You can use Application Signals to create *service level objectives* for the services for your critical business operations or dependencies. By creating SLOs on these services, you will be able to track them on the SLO dashboard, giving you an at-a-glance view of your most important operations.

In addition to creating a quick view your operators can use to see the current status of critical operations, you can use SLOs to track the longer-term performance of your services, to make sure that they are meeting your expectations. If you have service level agreements with customers, SLOs are a great tool to make sure that they are met.

Assessing your services' health with SLOs starts with setting clear, measurable objectives based on key performance metrics— *service level indicators (SLIs)*. An SLO tracks the SLI performance against the threshold and goal that you set, and reports how far or how close your application performance is to the threshold.

Application Signals helps you set SLOs on your key performance metrics. Application Signals automatically collects `Latency` and `Availability` metrics for every service and operation that it discovers, and these metrics are often ideal to use as SLIs. With the SLO creation wizard, you can use these metrics for your SLOs. You can then track the status of all of your SLOs with the Application Signals dashboards.

You can set SLOs on specific operations or dependencies that your service calls or uses. You can use any CloudWatch metric or metric expression as an SLI, in addition to using the `Latency` and `Availability` metrics.

Creating SLOs is very important for getting the most benefit from CloudWatch Application Signals. After you create SLOs, you can view their status in the Application Signals console to quickly see which of your these critical services and operations are performing well and which are unhealthy. Having SLOs to track provides the following major benefits:

- It is easier for your service operators to see the current operational health of critical services as measured against the SLI. Then they can quickly triage and identify unhealthy services and operations.
- You can track your service performance against measurable business goals over longer periods of time.

By choosing what to set SLOs on, you are prioritizing what is important to you. The Application Signals dashboards automatically present information about what you have prioritized.

When you create an SLO, you can also choose to create CloudWatch alarms at the same time to monitor the SLOs. You can set alarms that monitor for breaches of the threshold, and also for warning levels. These alarms can automatically notify you if the SLO metrics are breaching the threshold that you set, or if they are nearing a warning threshold. For example, an SLO nearing its warning threshold can let you know that your team might need to slow down churn in the application to make sure that long-term performance goals are met.

Topics

- SLO concepts
- Calculate error budget and attainment for period-based SLOs
- Calculate error budget and attainment for request-based SLOs
- Calculate burn rates and optionally set burn rate alarms
- Create an SLO
- Use SLO recommendations
- View and triage SLO status
- Edit an existing SLO
- Delete an SLO

SLO concepts

An SLO includes the following components:

- A *service level indicator (SLI)*, which is a key performance metric that you specify. It represents the desired level of performance for your application. Application Signals automatically collects the key metrics `Latency` and `Availability` for the services and operations that it discovers, and these can often be ideal metrics to set SLOs for.

  You choose the threshold to use for your SLI. For example, 200ms for latency.
- A *goal* or *attainment goal*, which is the percentage of time or requests that the SLI is expected to meet the threshold over each time *interval*. The time intervals can be as short as hours or as long as a year.

  Intervals can be either calendar intervals or rolling intervals.

  - *Calendar intervals* are aligned with the calendar, such as an SLO that is tracked per month. CloudWatch automatically adjusts health, budget, and attainment numbers based on the

number of days in a month. Calendar intervals are better suited for business goals that are measured on a calendar-aligned basis.

- *Rolling intervals* are calculated on a rolling basis. Rolling intervals are better suited for tracking recent user experience of your application.

- The *period* is a shorter length of time, and many periods make up an interval. The application's performance is compared to the SLI during each period within the interval. For each period, the application is determined to have either achieved or not achieved the necessary performance.

For example, a goal of 99% with a calendar interval of one day and a period of 1 minute means that the application must meet or achieve the success threshold during 99% of the 1-minute periods during the day. If it does, then the SLO is met for that day. The next day is a new evaluation interval. The application must meet or achieve the success threshold during 99% of the 1-minute periods during the second day to meet the SLO for that second day.

An SLI can be based on one of the new standard application metrics collected by Application Signals. Alternatively, it can be any CloudWatch metric or metric expression. The standard application metrics that you can use for an SLI are `Latency` and `Availability`. `Availability` represents the successful responses divided by the total requests. It is calculated as **(1 - Fault Rate)*100**, where Fault responses are `5xx` errors. Success responses are responses without a 5XX error. 4XX responses are treated as successful.

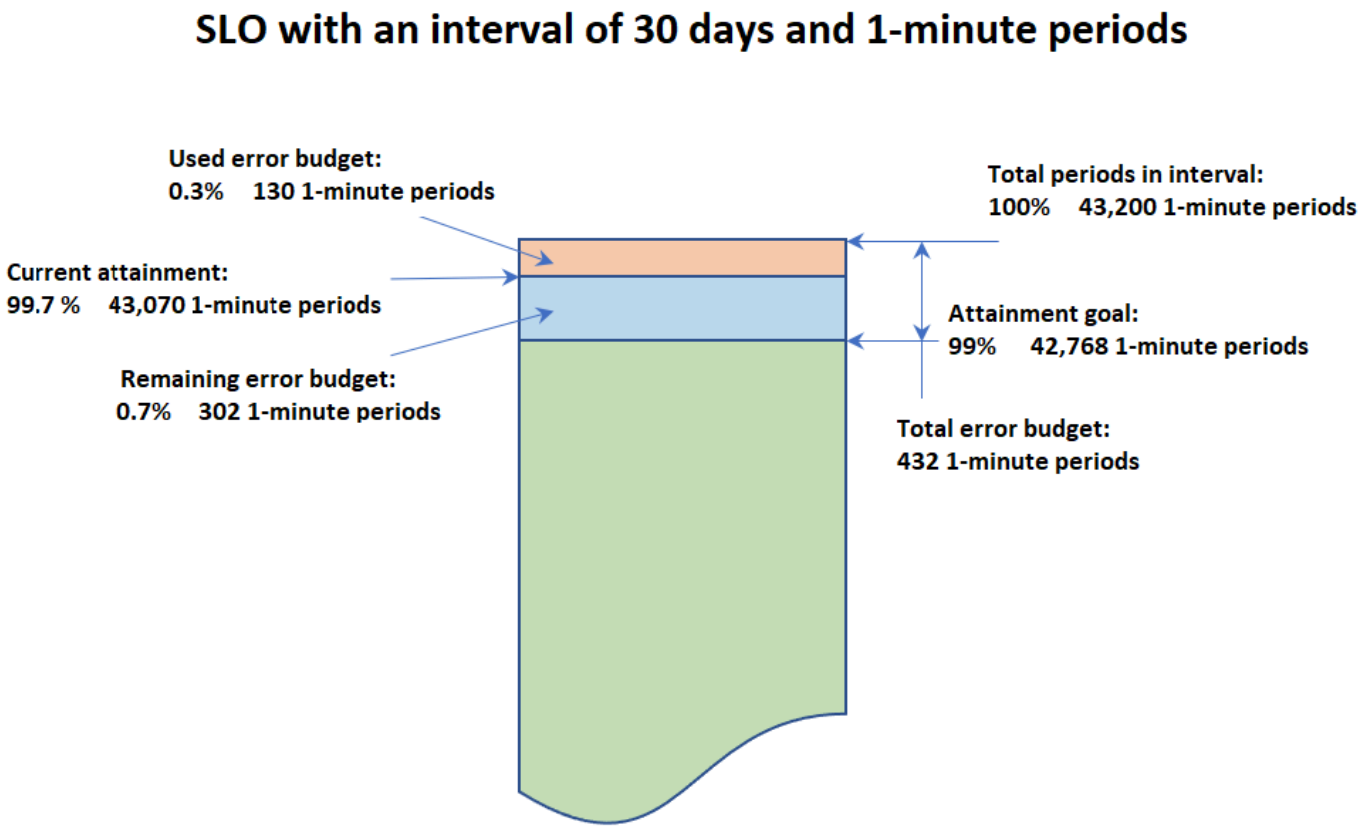
In addition to creating SLOs on a single operation or on all operations of a service, you can create *composite SLOs* that monitor a subset of operations for a service. Composite SLOs aggregate the `Availability` metric across multiple operations, giving you a unified view of reliability for a group of related operations. You can select between 2 and 20 operations to include in a composite SLO. For more information, see Create a composite SLO on multiple operations.

Calculate error budget and attainment for period-based SLOs

When you view information about an SLO, you see its current health status and its *error budget*. The error budget is the amount of time within the interval that can breach the threshold but still let the SLO be met. The *total error budget* is the total amount of breaching time that can be tolerated through the entire interval. The *remaining error budget* is the remaining amount of breaching time that can be tolerated during the current interval. This is after the amount of breaching time that has already happened has been subtracted from the total error budget.

The following figure illustrates the attainment and error budget concepts for a goal with a 30-day interval, 1-minute periods, and a 99% attainment goal. 30 days includes 43,200 1-minute periods.

99% of 43,200 is 42,768, so 42,768 minutes during the month must be healthy for the SLO to be met. So far in the current interval, 130 of the 1-minute periods were unhealthy.



Determine success within each period

Within each period, the SLI data is aggregated into a single data point based on the statistic used for the SLI. This data point represents the entire length of the period. That single data point is compared to the SLI threshold to determine if the period is healthy. Seeing unhealthy periods during the current time range on the dashboard can alert your service operators that the service needs to be triaged.

If the period is determined to be unhealthy, the entire length of the period is counted as failed against the error budget. Tracking the error budget lets you know whether the service is achieving the performance that you want over a longer period of time.

Time window exclusions

Time window exclusions is a block of time with a defined start and end date. This time period is excluded from the SLO's performance metrics and you can schedule one-time or recurring time exclusions windows. For example, scheduled maintenance.

Note

- For period-based SLOs, SLI data in the exclusion window is considered as non-breaching.
- For request-based SLOs, all good and bad requests in the exclusion window are excluded.
- When an interval for a request-based SLO is completely excluded, a default attainment rate metric of 100% is published.
- You can only specify time windows with a start date in the future.

Calculate error budget and attainment for request-based SLOs

After you have created an SLO, you can retrieve error budget reports for it. An *error budget* is the amount of requests that your application can be non-compliant with the SLO's goal, and still have your application meet the goal. For a request-based SLO, the remaining error budget is dynamic and can increase or decrease, depending on the ratio of good requests to total requests

The following table illustrates the calculation for a request-based SLO with an interval of 5 days and 85% attainment goal. In this example, assume there is no traffic before Day 1. The SLO did not meet the goal on Day 10.

Note

For request-based SLOs, `TotalRequestCountPerMinute` and `BadRequestCountPerMinute` are emitted as additional metrics compared to period-based SLO metrics. These metrics are provided for observability purposes and are not used as inputs to attainment-rate calculations.
Because these metrics are generated from periodically evaluated metric data, their values might occasionally differ from expected request counts due to metric publication timing or delays. Such discrepancies do not impact SLO attainment calculations, which are computed independently of these emitted per-minute metrics.

Time	Total requests	Bad requests	Accumulative total requests in last 5 days	Accumulative total good requests in last 5 days	Request-based attainment	Total budget requests
Day 1	10	1	10	9	9/10 = 90%	1.5
Day 2	5	1	15	13	13/15=86%	2.3
Day 3	1	1	16	13	13/16=81%	2.4
Day 4	24	0	40	37	37/40=92%	6.0
Day 5	20	5	60	52	52/60=87%	9.0
Day 6	6	2	56	47	47/56=84%	8.4
Day 7	10	3	61	50	50/61=82%	9.2
Day 8	15	6	75	59	59/75=79%	11.3
Day 9	12	1	63	46	46/63=73%	9.5
Day 10		5	57	40	40/57=70%	8.5
Final attainment for last 5 days					**70%**	

Calculate burn rates and optionally set burn rate alarms

You can use Application Signals to calculate the *burn rates* for your service level objectives. A burn rate is a metric that indicates how fast the service is consuming the error budget, relative to the attainment goal of the SLO. It's expressed as a multiple factor of the baseline error rate.

The burn rate is calculated according to the *baseline error rate*, which depends on the attainment goal. The attainment goal is the percentage of either healthy time periods or successful requests

that must be achieved to meet the SLO goal. The baseline error rate is (100% - attainment goal percentage), and this number would use up the exact complete error budget at the end of the SLO's time interval. So an SLO with an attainment goal of 99% would have a baseline error rate of 1%.

Monitoring the burn rate tells us how far off you are from the baseline error rate. Again taking the example of an attainment goal of 99%, the following is true:

- **Burn rate = 1**: If the burn rate remains exactly at the baseline error rate all the time, you meet exactly the SLO goal.
- **Burn rate < 1**: If the burn rate is lower than the baseline error rate, you are on track to exceed the SLO goal.
- **Burn rate > 1**: If the burn rate is higher than baseline error rate, you have a chance to fail the SLO goal.

When you create burn rates for your SLOs, you can also choose to create CloudWatch alarms at the same time to monitor the burn rates. You can set a threshold for the burn rates and the alarms can automatically notify you if the burn rate metrics are breaching the threshold that you set. For example, a burn rate nearing its threshold can let you know that the SLO is burning through the error budget faster than your team can tolerate and your team might need to slow down churn in the application to make sure that long-term performance goals are met.

Creating alarms incurs charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Calculate the burn rate

To calculate the burn rate, you must specify a *look-back window*. The look-back window is the time duration over which to measure the error rate.

```
burn rate = error rate over the look-back window / (100% - attainment goal)
```

> **Note**
>
> When there is no data for the burn rate period, Application Signals calculates the burn rate based on attainment.

The error rate is calculated as the ratio of the number of bad events over the total number of events during the burn rate window:

- For period-based SLOs, error rate is calculated as bad periods divided by total periods. The total periods represents the entirety of periods during the look-back window.
- For request-based SLOs, this is a measure of bad requests divided by total requests. The total number of requests is the number of requests during the look-back window.

The look-back window must be a multiple of the SLO period time, and must be less than the SLO interval.

Determine the appropriate threshold for a burn rate alarm

When you configure a burn rate alarm, you need to choose a value for the burn rate as the alarm threshold. The value for this threshold depends on the SLO interval length and look-back window, and depends on the which method or mental model that your team wants to adopt. There are two main methods available for determining the threshold.

Method 1: Determine the percentage of estimated total error budget your team is willing to burn in the look-back window.

If you want to get alarmed when X% of the estimated error budget is spent within the last burn rate look-back hours, the burn rate threshold is the following:

```
burn rate threshold = X% * SLO interval length / look-back window size
```

For example, 5% of a 30-day (720-hour) error budget spent over one hour requires a burn rate of `5% * 720 / 1 = 36`. Therefore, if the burn rate look-back window is1 hour, you set the burn rate threshold to be 36.

You can use the CloudWatch console to create burn rate alarms using this method. You can specify the number X, and the threshold is determined using the above formula.

The SLO interval length is determined based on the SLO interval type:

- For SLOs with a rolling interval, it's the length of the interval in hours.
- For SLOs with a calendar-based interval:
  - If the unit is days or weeks, it's the length of the interval in hours.

- If the unit is a month, 30 days is taken as the estimated length and convert it to hours.

Method 2: Determine the time unitl budget exhaustion for the next interval

To have the alarm notify you when the current error rate in the most recent look-back window indicates that the time until budget exhaustion is less than X hours away (assuming the budget remaining is currently 100%), you can use the following formula to determine the burn rate threshold.

```
burn rate threshold = SLO interval length / X
```

Note that the time until budget exhaustion (X) in the above formula assumes that the total budget remaining is currently 100%, and therefore it does not take into account the amount of budget that has already been burnt in this interval. You can also think of it as the time till budget exhaustion for the next interval.

Walkthroughs for burn rate alarms

As an example, let's take an SLO with a 28-day rolling interval. Setting a burn rate alarm for this SLO involves two steps:

1. Set the burn rate and the look-back window.
2. Crete a CloudWatch alarm that monitors the burn rate.

To get started, determine how much of the total error budget the service is willing to burn through within a specific time frame. In other words, state your objective by using this sentence: "I want to get alerted when X% of my total error budget is consumed within M minutes."

For example, you might want to set the objective to be alerted when 2% of the total error budget is consumed within 60 minutes.

To set the burn rate, you first define the look-back window. The look-back windows is M, which in this example is 60 minutes.

Next, you create the CloudWatch alarm. When you do so, you must specify a threshold for the burn rate. If burn rate exceeds this threshold, the alarm will notify you. To find the threshold,use the following formula:

```
burn rate threshold = X% * SLO interval length/ look-back window size
```

In this example, X is 2 because you want to be alerted if 2% of the error budget is consumed within 60 minutes. The interval length is 40,320 minutes (28 days), and 60 minutes is the look-back window, so the answer is:

```
burn rate threshold = 2% * 40,320 / 60 = 13.44.
```

In this example, you would set 13.44 as the alarm threshold.

Multiple alarms with different windows

By setting up alarms on multiple look-back windows, you can quickly detect sharp error rate increases with the short window and at the same time detect smaller error rate increases that eventually deplete the error budget if they remain unnoticed.

Additionally, you could set a *composite alarm* on a burn rate with long window and on a burn rate with a short window (1/12th of the long window), and be informed only when both of the burn rates breach a threshold. This way, you can make sure that you get alerted only for situations that are still happening. For more information about composite alarms in CloudWatch, see Create a composite alarm.

> **Note**
>
> You can set a metric alarm on a burn rate when you create the burn rate. To set a compoaite alarm on multiple burn rate alarms, you must use the instructions in Create a composite alarm.

One composite alarm strategy recommended in the Google Site Reliability Engineering workbook includes three composite alarms:

- One composite alarm that watches a pair of alarms, one with a one-hour window and one with a five-minute window.
- A second composite alarm that watches a pair of alarms, one with a six-hour window and one with a 30-minute window.
- A third composite alarm that watches a pair of alarms, one with a three-day window and one with a six-hour window.

The steps to do this set up are the following:

1. Create five burn rates, with windows of five minutes, 30 minutes, one hour, six hours, and three days.
2. Create the following three pairs of CloudWatch alarms. Each pair includes one long window and one short window that is 1/12th of the long window, and the thresholds are determined by using the steps in Determine the appropriate threshold for a burn rate alarm. When you calculate the threshold for each alarm in the pair, use the longer look-back window of the pair in your calculation.
   - Alarms on the 1-hour and 5-minute burn rates (the threshold is determined by 2% of the total budget)
   - Alarms on the 6-hour and 30-minute burn rates (the threshold is determined by 5% of the total budget)
   - Alarms on the 3-day and 6-hour burn rates (the threshold is determined by 10% of the total budget)
3. For each of these pairs, create a composite alarm to get alerted when both of the individual alarms go into ALARM state. For more information about creating composite alarms, see Create a composite alarm.

   For example, if your alarms for the first pair (one-hour window and five-minute window) are named `OneHourBurnRate` and `FiveMinuteBurnRate`, the CloudWatch composite alarm rule would be `ALARM(OneHourBurnRate) AND ALARM(FiveMinuteBurnRate)`

The previous strategy is possible only for SLOs with interval length of at least three hours. For SLOs with shorter interval lengths, CloudWatch recommend that you start with one pair of burn rate alarms where one alarm has a look-back window that is 1/12th of the look-back window of the other alarm. Then set a composite alarm on this pair.

Create an SLO

We recommend that you set both latency and availability SLOs on your critical applications. These metrics collected by Application Signals align with common business goals.

You can also set SLOs on any CloudWatch metric or any metric math expression that results in a single time series.

The first time that you create an SLO in your account, CloudWatch automatically creates the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role in your account, if it doesn't already exist. This service-linked role allows CloudWatch to collect CloudWatch Logs data,

X-Ray trace data, CloudWatch metrics data, and tagging data from applications in your account. For more information about CloudWatch service-linked roles, see Using service-linked roles for CloudWatch.

When you create an SLO, you specify whether it is a *period-based SLO* or a *request-based SLO*. Each type of SLO has a different way of evaluating your application's performance against its attainment goal.

- A *period-based SLO* uses defined *periods* of time within a specified total time interval. For each period of time, Application Signals determines whether the application met its goal. The attainment rate is calculated as the `number of good periods/number of total periods`.

  For example, for a period-based SLO, meeting an attainment goal of 99.9% means that within your interval, your application must meet its performance goal during at least 99.9% of the time periods.
- A *request-based SLO* doesn't use pre-defined periods of time. Instead, the SLO measures `number of good requests/number of total requests` during the interval. At any time, you can find the ratio of good requests to total requests for the interval up to the time stamp that you specify, and measure that ratio against the goal set in your SLO.

Topics

- Create a period-based SLO
- Create a request-based SLO
- Create an SLO on an app monitor
- Create an SLO on a canary
- Create a composite SLO on multiple operations

Create a period-based SLO

Use the following procedure to create a period-based SLO.

To create a period-based SLO

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.

3. Choose **Create SLO**.
4. For **Set Service Level Indicator (SLI)**, do one of the following:
   - To set the SLO on a service operation, all operations, or the dependency of a service, using either of the standard application metrics `Latency` or `Availability`:
     a. For **Type**, choose **Service**.
     b. Select an account that this SLO will monitor.
     c. Select the service that this SLO will monitor.
     d. For **Type**, choose one of the following:
        - **Service Operations** — to create an SLO on a service operation, all operations, or a subset of operations.
        - **Service Dependency** — to create an SLO on a dependency of the service.
     e. If you chose **Service Operations**, select the operation that this SLO will monitor. To create a service-level SLO that monitors the overall health of your service across all operations, select **All Operations**. Otherwise, select a specific operation to monitor.

        To create an SLO that monitors a subset of operations, see Create a composite SLO on multiple operations.
     f. If you chose **Service Dependency**, do the following:
        i. Under **Select an operation**, select one specific operation or select **All operations** to use the metrics from all operations of this service that calls a dependency.
        ii. Under **Select a dependency**, search and select the required dependency for which you want to measure the reliability.

            After you select the dependency, you can view the updated graph and historical data based on the dependency.
     g. For **Select a calculation method**, choose **Periods**.

        The **Select service** and **Select operation** drop-downs are populated by services and operations that have been active within the past 24 hours.
     h. Choose either **Availability** or **Latency** and then set the threshold.
   - To set the SLO on any CloudWatch metric or a CloudWatch metric math expression:
     a. For **Type**, choose **CloudWatch Metric**.

b. Choose **Select CloudWatch metric**.

The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

After you select the metric that you want, choose the **Graphed metrics** tab and select the **Statistic** and **Period** to use for the SLO. Then choose **Select metric**.

For more information about these screens, see Graph a metric and Add a math expression to a CloudWatch graph.

c. For **Select a calculation method**, choose **Periods**.

d. For **Set condition**, select a comparison operator and threshold for the SLO to use as the indicator of success.

5. If you selected **Service** in step 4, set the period length for this SLO.
6. Enter a name for the SLO. Including the name of a service or operation, along with appropriate keywords such as latency or availability, will help you quickly identify what the SLO status indicates during triage.
7. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see SLO concepts.
8. (Optional) For **Set SLO burn rates** do the following:

   - Set the length (in minutes) of the look-back window for the burn rate. For information about how to choose this length, see Walkthroughs for burn rate alarms.
   - To create more burn rates for this SLO, choose **Add more burn rates** and set the look-back window for the additional burn rates.

9. (Optional) Create burn rate alarms by doing the following:

   - Under **Set burn rate alarms** select the check box for each burn rate that you want to create an alarm for. For each of these alarms, do the following:
     - Specify the Amazon SNS topic to use for notifications when the alarm goes into ALARM state.
     - Either set a burn rate threshold or specify the percentage of the estimated total budget burnt in the last look-back window you want to stay below. If you set the percentage of estimated total budget burned, the burn rate threshold is calculated for you and used in the alarm. To either decide what threshold to set or to understand how this option is used

to compute the burn rate threshold, see Determine the appropriate threshold for a burn rate alarm.

10. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.

    a. CloudWatch alarms can use Amazon SNS to proactively notify you if an application is unhealthy based on its SLI performance.

    To create an alarm, select one of the alarm check boxes and enter or create the Amazon SNS topic to use for notifications when the alarm goes into ALARM state. For more information about CloudWatch alarms, see Using Amazon CloudWatch alarms. Creating alarms incurs charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

    b. If you set a warning threshold, it appears in Application Signals screens to help you identify SLOs that are in danger of being unmet, even if they're currently healthy.

    To set a warning threshold, enter the threshold value in **Warning threshold**. When the SLO's error budget is lower than the warning threshold, the SLO is marked with **Warning** in several Application Signals screens. Warning thresholds also appear on error budget graphs. You can also create an **SLO warning alarm** that's based on the warning threshold.

11. (Optional) For **Set SLO time window exclusion**, do the following:

    - Under **Exclude time window**, set the time window to be excluded from the SLO performance metrics.

      You can choose **Set time window** and enter the **Start window** for every hour or month or you can choose **Set time window with CRON** and enter the CRON expression.
    - Under **Repeat**, set if this time window exclusion is recurring or not.
    - (Optional) Under **Add reason**, you can choose to enter a reason for the time window exclusion. For example, scheduled maintenance.
    - Select **Add time window** to add upto 10 time exclusion windows.

12. To add tags to this SLO, choose the **Tags** tab and then choose **Add new tag**. Tags can help you manage, identify, organize, search for, and filter resources. For more information about tagging, see Tagging your AWS resources.

> **Note**
>
> If the application this SLO is related to is registered in AWS Service Catalog AppRegistry, you can use the `awsApplication` tag to associate this SLO with that application in AppRegistry. For more information, see What is AppRegistry?

13. Choose **Create SLO**. If you also chose to create one or more alarms, the button name changes to reflect this.

Create a request-based SLO

Use the following procedure to create a request-based SLO.

To create a request-based SLO

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. For **Set Service Level Indicator (SLI)**, do one of the following:
   - To set the SLO on a service operation, all operations, or the dependency of a service, using either of the standard application metrics `Latency` or `Availability`:
     a. For **Type**, choose **Service**.
     b. Select the service that this SLO will monitor.
     c. For **Type**, choose one of the following:
        - **Service Operations** — to create an SLO on a service operation, all operations, or a subset of operations.
        - **Service Dependency** — to create an SLO on a dependency of the service.
     d. If you chose **Service Operations**, select the operation that this SLO will monitor. To create a service-level SLO that monitors the overall health of your service across all operations, select **All Operations**. Otherwise, select a specific operation to monitor.

        To create an SLO that monitors a subset of operations, see Create a composite SLO on multiple operations.
     e. If you chose **Service Dependency**, do the following:

i. Under **Select an operation**, select one specific operation or select **All operations** to use the metrics from all operations of this service that calls a dependency.

ii. Under **Select a dependency**, search and select the required dependency for which you want to measure the reliability.

After you select the dependency, you can view the updated graph and historical data based on the dependency.

f. For **Select a calculation method**, choose **Requests**.

g. The **Select service** and **Select operation** drop-downs are populated by services and operations that have been active within the past 24 hours.

h. Choose either **Availability** or **Latency**. If you choose **Latency**, set the threshold.

- To set the SLO on any CloudWatch metric or a CloudWatch metric math expression:

a. For **Type**, choose **CloudWatch Metric**.

b. For **Define target requests**, do the following:

i. Choose whether you want to measure **Good Requests** or **Bad Requests**.

ii. Choose **Select CloudWatch metric**. This metric will be the numerator of the ratio of target requests to total requests. If you use a latency metric, use the **Trimmed count (TC)** statistics. If the threshold is 9 ms and you're using the less than (<) comparison operator, then use threshold TC (:threshold - 1). For more information about TC, see Syntax.

The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

c. For **Define total requests**, choose the CloudWatch metric that you want to use for the source. This metric will be the denominator of the ratio of target requests to total requests.

The **Select metric** screen appears. Use the **Browse** or **Query** tabs to find the metric you want, or create a metric math expression.

After you select the metric that you want, choose the **Graphed metrics** tab and select the **Statistic** and **Period** to use for the SLO. Then choose **Select metric**.

If you use a latency metric which emits one data point per request, use the **Sample count statistics** to count the number of total requests.

For more information about these screens, see Graph a metric and Add a math expression to a CloudWatch graph.

5. Enter a name for the SLO. Including the name of a service or operation, along with appropriate keywords such as latency or availability, will help you quickly identify what the SLO status indicates during triage.
6. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see SLO concepts.
7. (Optional) For **Set SLO burn rates** do the following:
   - Set the length (in minutes) of the look-back window for the burn rate. For information about how to choose this length, see Walkthroughs for burn rate alarms.
   - To create more burn rates for this SLO, choose **Add more burn rates** and set the look-back window for the additional burn rates.
8. (Optional) Create burn rate alarms by doing the following:
   - Under **Set burn rate alarms** select the check box for each burn rate that you want to create an alarm for. For each of these alarms, do the following:
     - Specify the Amazon SNS topic to use for notifications when the alarm goes into ALARM state.
     - Either set a burn rate threshold or specify the percentage of the estimated total budget burnt in the last look-back window you want to stay below. If you set the percentage of estimated total budget burned, the burn rate threshold is calculated for you and used in the alarm. To either decide what threshold to set or to understand how this option is used to compute the burn rate threshold, see Determine the appropriate threshold for a burn rate alarm.
9. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.
   a. CloudWatch alarms can use Amazon SNS to proactively notify you if an application is unhealthy based on its SLI performance.

      To create an alarm, select one of the alarm check boxes and enter or create the Amazon SNS topic to use for notifications when the alarm goes into ALARM state. For more information about CloudWatch alarms, see Using Amazon CloudWatch alarms. Creating alarms incurs charges. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

b. If you set a warning threshold, it appears in Application Signals screens to help you identify SLOs that are in danger of being unmet, even if they're currently healthy.

   To set a warning threshold, enter the threshold value in **Warning threshold**. When the SLO's error budget is lower than the warning threshold, the SLO is marked with **Warning** in several Application Signals screens. Warning thresholds also appear on error budget graphs. You can also create an **SLO warning alarm** that's based on the warning threshold.

10. (Optional) For **Set SLO time window exclusion**, do the following:

    - Under **Exclude time window**, set the time window to be excluded from the SLO performance metrics.

      You can choose **Set time window** and enter the **Start window** for every hour or month or you can choose **Set time window with CRON** and enter the CRON expression.
    - Under **Repeat**, set if this time window exclusion is recurring or not.
    - (Optional) Under **Add reason**, you can choose to enter a reason for the time window exclusion. For example, scheduled maintenance.
    - Select **Add time window** to add upto 10 time exclusion windows.

11. To add tags to this SLO, choose the **Tags** tab and then choose **Add new tag**. Tags can help you manage, identify, organize, search for, and filter resources. For more information about tagging, see Tagging your AWS resources.

    > **Note**
    >
    > If the application this SLO is related to is registered in AWS Service Catalog AppRegistry, you can use the `awsApplication` tag to associate this SLO with that application in AppRegistry. For more information, see What is AppRegistry?

12. Choose **Create SLO**. If you also chose to create one or more alarms, the button name changes to reflect this.

Create an SLO on an app monitor

You can create SLOs to monitor the performance of your CloudWatch RUM app monitors. This allows you to track real user experience metrics and make sure your web and mobile applications meet performance goals. SLOs on app monitors use request-based evaluation, which measures the ratio of good requests to total requests.

To create an SLO on an app monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. For **Set Service Level Indicator (SLI)**, choose **RUM AppMonitor**.
5. Select the app monitor that this SLO will monitor from the dropdown list. The list shows the app monitor name along with the supported platform (Web, iOS, or Android).
6. (Optional) Select a specific page or screen to monitor. If you don't select a page, the SLO will monitor all pages for the app monitor.
7. For **Select metric**, choose the metric to use for the SLI. The available metrics depend on the platform:
   - For web applications: `PerformanceNavigationDuration`, `JSErrorCount`, `Http4xxCount`, and `Http5xxCount`
   - For mobile applications (iOS and Android): `ScreenLoadTime`, `CrashCount`, `Http4xxCount`, and `Http5xxCount`
8. For **Set condition**, select a comparison operator and threshold for the SLO to use as the indicator of success.
9. Enter a name for the SLO. Including the app monitor name and appropriate keywords will help you quickly identify what the SLO status indicates during triage.
10. Set the **interval** and **attainment goal** for the SLO. For more information, see SLO concepts.
11. (Optional) Configure burn rates and alarms as needed. For more information, see Calculate burn rates and optionally set burn rate alarms.
12. (Optional) Set time window exclusions if needed.
13. (Optional) Add tags to help organize and identify this SLO.
14. Choose **Create SLO**.

Create an SLO on a canary

You can create SLOs to monitor the performance of your CloudWatch Synthetics canaries. This allows you to track synthetic monitoring results and make sure your endpoints and APIs meet availability and performance goals. SLOs on canaries use period-based evaluation, where each canary run is treated as a discrete evaluation period.

To create an SLO on a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. For **Set Service Level Indicator (SLI)**, choose **Synthetics Canary**.
5. Select the canary that this SLO will monitor from the dropdown list.
6. For **Select metric**, choose either `SuccessPercent` or `Duration`:
   - `SuccessPercent` measures the percentage of successful canary runs
   - `Duration` measures how long each canary run takes to complete
7. For **Set condition**, select a comparison operator and threshold for the SLO to use as the indicator of success.
8. Enter a name for the SLO. Including the canary name and appropriate keywords will help you quickly identify what the SLO status indicates during triage.
9. Set the **interval** and **attainment goal** for the SLO. For more information, see SLO concepts.
10. (Optional) Configure burn rates and alarms as needed. For more information, see Calculate burn rates and optionally set burn rate alarms.
11. (Optional) Set time window exclusions if needed.
12. (Optional) Add tags to help organize and identify this SLO.
13. Choose **Create SLO**.

Create a composite SLO on multiple operations

You can create a composite SLO that monitors the `Availability` metric across a subset of operations for a service. This is useful when you want to track the reliability of a group of related operations together, rather than monitoring a single operation or all operations.

Composite SLOs support both period-based and request-based evaluation. You can select between 2 and 20 operations to include. There are two ways to select operations:

- *Explicit selection* — Manually pick individual operations from the dropdown.
- *Pattern matching* — Use a prefix or regular expression to automatically match operations by name.

> **Note**
>
> Composite SLOs support only the `Availability` metric. The `Latency` metric is not available for composite SLOs.

To create a composite SLO

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose **Create SLO**.
4. For **Set Service Level Indicator (SLI)**, for **Type**, choose **Service**.
5. Select the service that this SLO will monitor.
6. For **Type**, choose **Service Operations**.
7. Select the operations to include in this composite SLO. Do one of the following:
   - To manually select operations, choose multiple operations from the **Operation** dropdown. You can select between 2 and 20 operations.

     The selected operations appear as tokens below the dropdown. You can remove an operation by choosing the dismiss icon on its token.
   - To select operations by pattern, select the **Use pattern matching** checkbox. Then do the following:

     a. For **Pattern type**, choose either **Prefix** or **Regular expression**.

        - **Prefix** matches all operations whose names start with the text you enter. For example, entering `Invoke` matches operations named `InvokeFunction`, `InvokeAsync`, and so on.
        - **Regular expression** matches all operations whose names match the regex pattern you enter. For example, entering `^Invoke.*` matches the same operations as the prefix example.

     b. Enter the pattern in the **Pattern** field. The console displays the matched operations as tokens below the field so you can verify the results.

   After you select operations, the metric is automatically set to **Availability**.
8. For **Select a calculation method**, choose either **Periods** or **Requests**.

9. If you selected **Periods**, set the period length and the availability threshold for this SLO.
10. Enter a name for the SLO, or use the auto-generated name. The auto-generated name includes the service name and the word "composite" to help you identify it.
11. Set the **interval** and **attainment goal** for the SLO. For more information about intervals and attainment goals and how they work together, see SLO concepts.
12. (Optional) Configure burn rates and alarms as needed. For more information, see Calculate burn rates and optionally set burn rate alarms.
13. (Optional) Set one or more CloudWatch alarms or a warning threshold for the SLO.
14. (Optional) Set time window exclusions if needed.
15. (Optional) Add tags to help organize and identify this SLO.
16. Choose **Create SLO**.

Use SLO recommendations

Application Signals can provide recommendations for your SLO configuration based on historical metric data from the last 30 days. When you provide basic information about your service and the type of SLO you want to create, Application Signals analyzes your metric data and suggests optimal values for the metric threshold, SLO goal, and burn rate windows.

To receive SLO recommendations, you must provide the following information:

- Choose either **Service Operation** or **Service Dependency**:
  - For **Service Operation**, specify the service and operation
  - For **Service Dependency**, specify the service, operation (or all operations), and dependency
- The SLO evaluation type: either *period-based* or *request-based*
- The type of standard application metric: either `Latency` or `Availability`

Based on this information and your service's historical performance data, Application Signals recommends the following SLO configuration parameters:

- **Metric threshold** - The performance threshold for your SLI, calculated based on your service's actual performance over the last 30 days.
- **SLO goal** - The suggested attainment goal percentage that aligns with your service's historical reliability.

- **Burn rate windows** - Recommended look-back window durations for monitoring how quickly your service consumes its error budget.

You can accept the recommended values or adjust them based on your specific business requirements. The recommendations provide a data-driven starting point for configuring SLOs that reflect your service's actual performance characteristics.

View and triage SLO status

You can quickly see the health of your SLOs using either the **Service Level Objectives** or the **Services** options in the CloudWatch console. The **Services** view provides an at-a-glance view of the ratio of unhealthy services, calculated based on SLOs that you have set. For more information about using the **Services** option, see Monitor the operational health of your applications with Application Signals.

The **Service Level Objectives** view provides a macro view of your organization. You can see the met and unmet SLOs as a whole. This gives you a view of how many of your services and operations are performing to your expectations over longer periods of time, according to the SLIs that you chose.

To view all of your SLOs using the Service Level Objectives view

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.

   The **Service Level Objectives (SLO)** list appears.

   You can quickly see the current status of your SLOs in the **SLI status** column. To sort the SLOs so that all the unhealthy ones are at the top of the list, choose the **SLI status** column until the unhealthy SLOs are all at the top.

   The SLO table has the following default columns. You can adjust which columns are displayed by choosing the gear icon above the list. For more information about goals, SLIs, attainment, and intervals, see SLO concepts.

   - The name of the SLO.
   - The **Goal** column displays the percentage of periods during each interval that must successfully meet the SLI threshold for the SLO goal to be met. It also displays the interval length for the SLO.

- The **SLI status** displays whether the current operational state of the application is healthy or not. If any period during the currently selected time range was unhealthy for the SLO, then the **SLI status** displays **Unhealthy**.
- If this SLO is configured to monitor a dependency, the **Dependency** and **Remote Operation** columns will show the details about that dependency relationship.
- The **Ending attainment** is the attainment level achieved as of the end of the selected time range. Sort by this column to see the SLOs that are most in danger of not being met.
- The **Attainment delta** is the difference in attainment level between the start and end of the selected time range. A negative delta means that the metric is trending in a downward direction. Sort by this column to see the latest trends of the SLOs.
- The **Ending error budget (%)** is the percentage of total time in the period that can have unhealthy periods and still have the SLO be achieved successfully. If you set this to 5%, and the SLI is unhealthy in 5% or fewer of the remaining periods in the interval, the SLO is still achieved successfully.
- The **Error budget delta** is the difference in error budget between the start and end of the selected time range. A negative delta means that the metric is trending in a failing direction.
- The **Ending error budget (time)** is the amount of actual time in the interval that can be unhealthy and still have the SLO be achieved successfully. For example, if this is 14 minutes, then if the SLI is unhealthy for fewer than 14 minutes during the remaining interval, the SLO will still be achieved successfully.
- The **Ending error budget (requests)** is the amount of requests in the interval that can be unhealthy and still have the SLO be achieved successfully. For request-based SLOs, this value is dynamic and can fluctuate as the cumulative total number of requests changes over time.
- The **Service**, **Operation**, and **Type** columns display information about what service and operation this SLO is set for.

3. To see the attainment and error budget graphs for an SLO, choose the radio button next to the SLO name.

   The graphs at the top of the page display the **SLO attainment** and **Error budget** status. A graph about the SLI metric associated with this SLO is also displayed.

4. To further triage an SLO that is not meeting its goal, choose the service name, operation name, or dependency name associated with that SLO. You are taken to the details page where you can triage further. For more information, see View detailed service activity and operational health with the service detail page.

5. To change the time range of the charts and tables on the page, choose a new time range near the top of the screen.

Edit an existing SLO

Follow these steps to edit an existing SLO. When you edit an SLO, you can change only the threshold, interval, attainment goal, and tags. To change other aspects such as service, operation, or metric, create a new SLO instead of editing an existing one.

Changing part of an SLO core configuration, such as period or threshold, invalidates all the previous data points and assessments about attainment and health. It effectively deletes and re-creates the SLO.

> **Note**
>
> If you edit an SLO, alarms associated with that SLO are not automatically updated. You might need to update the alarms to keep them in sync with the SLO.

To edit an existing SLO

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions**, **Edit SLO**.
4. Make your changes, then choose **Save changes**.

Delete an SLO

Follow these steps to delete an existing SLO.

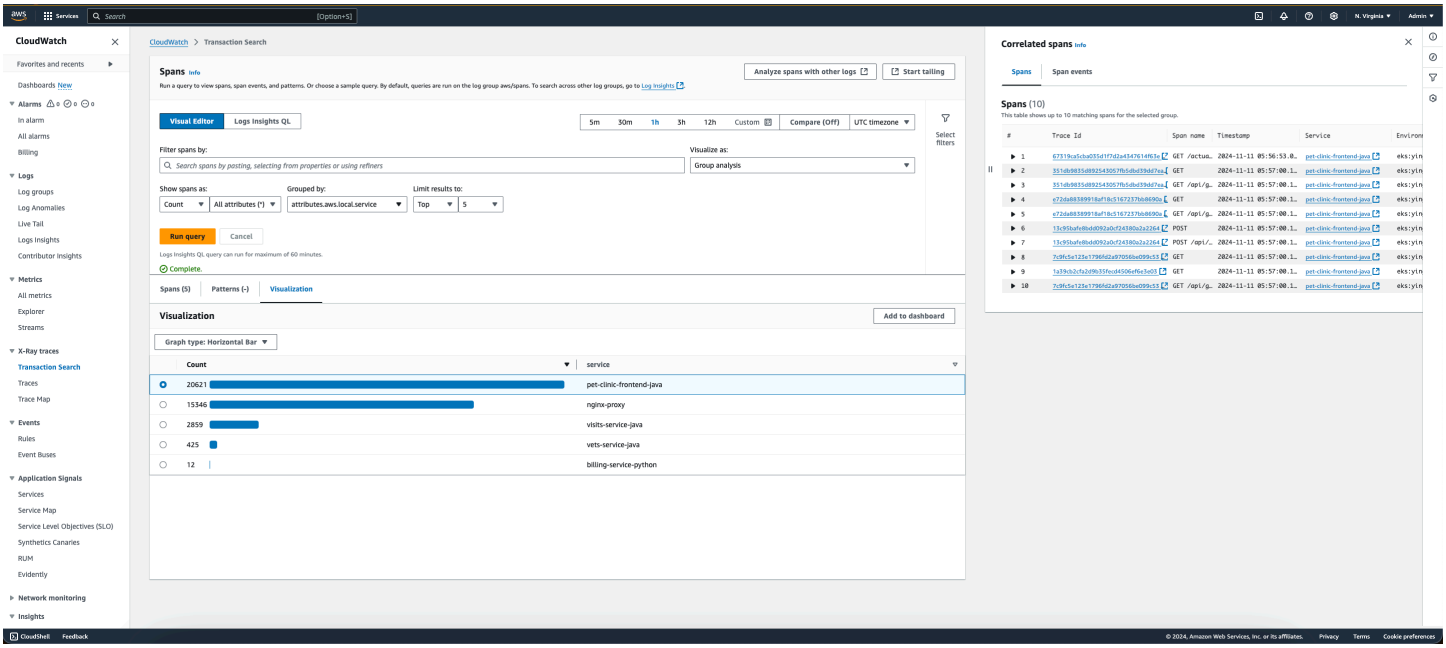
> **Note**
>
> When you delete an SLO, alarms associated with that SLO are not automatically deleted. You'll need to delete them yourself. For more information, see Managing alarms.

To delete an SLO

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Service Level Objectives (SLO)**.
3. Choose the radio button next to the SLO that you want to edit, and choose **Actions**, **Delete SLO**.
4. Choose **Confirm**.

Transaction Search

Transaction Search is an interactive analytics experience you can use to get complete visibility of your application transaction spans. Spans are the fundamental units of operation in a distributed trace and represent specific actions or tasks in an application or system. Every span records details about a particular segment of the transaction. These details include start and end times, duration, and associated metadata, which can include business attributes like customer IDs and order IDs. Spans are arranged in a parent-child hierarchy. This hierarchy forms a complete trace, mapping the flow of a transaction across different components or services.



Topics

- Benefits
- How it works
- Pricing

- Enable transaction search
- Spans
- Adding custom attributes
- Troubleshooting application issues

Benefits

The following are benefits of using Transaction Search:

Capture all spans

Ingest 100 percent of spans as structured logs in CloudWatch to get complete visibility. This prevents broken traces and provides you the ability to view large traces containing up to 10,000 spans for detailed insights.

Index spans as trace summaries

Index a percentage of spans as trace summaries in X-Ray to unlock end to end trace search and analytics.

Investigate transaction issues with free form analytics

Search all span attributes in the visual editor to identify the cause of issues in application transactions. This helps you answer questions about application performance and the impact end-users make based on their application transactions.

Send spans to the OpenTelemetry endpoint

Send spans to the OpenTelemetry endpoint for X-Ray traces. These spans are stored in the semantic convention format with W3C trace IDs.

> **Note**
>
> X-Ray traces automatically convert to the semantic convention format before they're stored in a log group called `aws/spans`. For more information, see The span log group.

Use CloudWatch Logs with spans

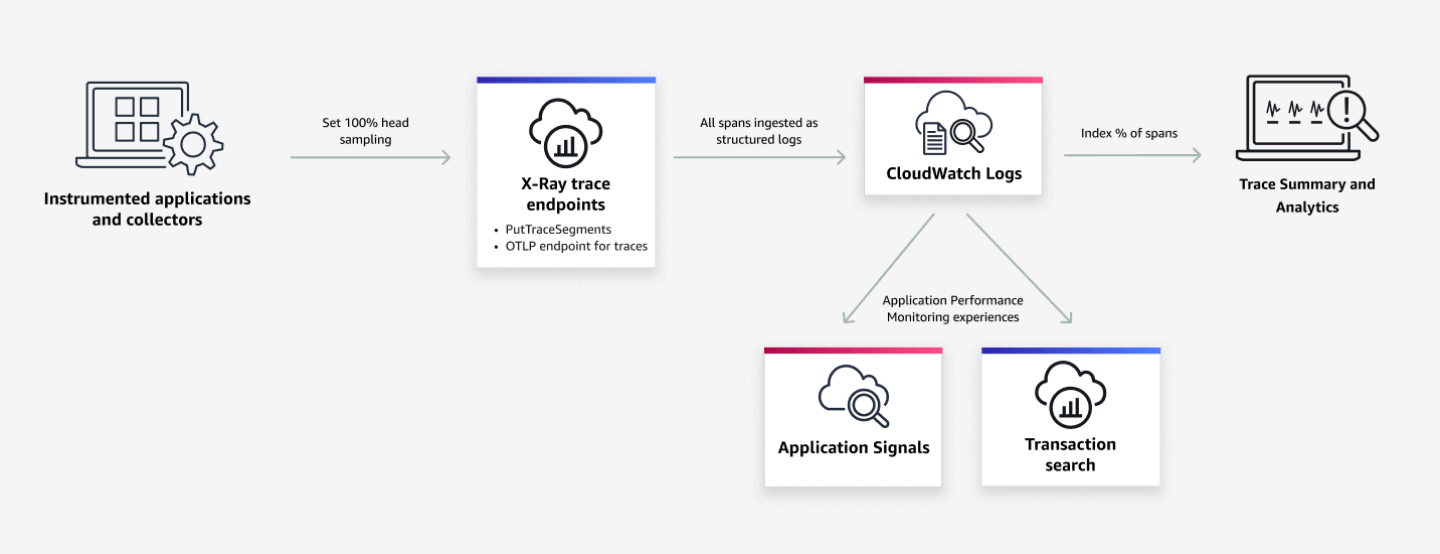
Use metric filters to extract custom metrics, subscription filters to forward data, and data masking to protect personally identifiable information.

Troubleshoot application issues

Access application dashboards, metrics, and topology when you enable Application Signals for all spans sent to CloudWatch.

How it works

When you enable Transaction Search, you unlock multiple capabilities, including features in Application Signals and CloudWatch Logs.



If you send traces to X-Ray, you can get started by enabling Transaction Search in the console or with the API. If you don't send traces to X-Ray, you can use the CloudWatch Application Signals that provides pre-packaged OpenTelemetry setup with AWS Distro for OpenTelemetry (ADOT), CloudWatch Agent, or use OpenTelemetry directly.

When you enable Transaction Search, spans sent to X-Ray are ingested in a log group called `aws/spans`. CloudWatch uses these spans to generate a curated application performance monitoring (APM) experience in CloudWatch Application Signals. This provides you the ability to search and analyze spans, as well as use CloudWatch Logs capabilities like anomaly and pattern detection. You can even use custom metric extraction . CloudWatch Application Signals provides you with a unified, application-centric view of your applications, services, and dependencies. It also helps you monitor and triage application health.

You can also explore spans using the interactive search and analytics experience in CloudWatch to answer any questions related to application performance or end-user impact with Transaction Search. Detect the impact on end users, find transactions in context of those issues using relevant attributes, such as customer name or order number. You can correlate transactions to business events, such as failed payments, and dive into interactions between application components to establish a root cause. With CloudWatch, you get complete application transaction coverage with correlated insights, helping you to accelerate mean time to resolution.

Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Enable transaction search

You can enable Transaction Search through the console or by using an API. Transaction search is configured for the entire account and switches all spans ingestion through X-Ray into cost effective collection mode using Amazon CloudWatch Pricing. By default you will also index 1% of ingested spans for free as trace summary for analysis, which is typically sufficient given you already have full end-to-end trace visibility on all ingested spans through Transaction Search.

Prerequisites

Before you can enable Transaction Search, you must create a role with the following permissions.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "TransactionSearchXRayPermissions",
      "Effect": "Allow",
      "Action": [
        "xray:GetTraceSegmentDestination",
        "xray:UpdateTraceSegmentDestination",
        "xray:GetIndexingRules",
        "xray:UpdateIndexingRule"
      ],
      "Resource": "*"
    },
    {
```

```
      "Sid": "TransactionSearchLogGroupPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutRetentionPolicy"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*",
        "arn:aws:logs:*:*:log-group:aws/spans:*"
      ]
    },
    {
      "Sid": "TransactionSearchLogsPermissions",
      "Effect": "Allow",
      "Action": [
        "logs:PutResourcePolicy",
        "logs:DescribeResourcePolicies"
      ],
      "Resource": "*"
    },
    {
      "Sid": "TransactionSearchApplicationSignalsPermissions",
      "Effect": "Allow",
      "Action": [
        "application-signals:StartDiscovery"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchApplicationSignalsCreateServiceLinkedRolePermissions",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "application-signals.cloudwatch.amazonaws.com"
        }
      }
    },
    {
      "Sid": "CloudWatchApplicationSignalsGetRolePermissions",
      "Effect": "Allow",
```

```
      "Action": "iam:GetRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/application-
signals.cloudwatch.amazonaws.com/AWSServiceRoleForCloudWatchApplicationSignals"
    },
    {
      "Sid": "CloudWatchApplicationSignalsCloudTrailPermissions",
      "Effect": "Allow",
      "Action": [
        "cloudtrail:CreateServiceLinkedChannel"
      ],
      "Resource": "arn:aws:cloudtrail:*:*:channel/aws-service-channel/
application-signals/*"
    }
  ]
}
```

Note

If you encrypt spans with a customer managed AWS KMS key, the role also needs the `kms:DescribeKey` permission for the key. If your role uses the `CloudWatchFullAccessV2` managed policy, it already has this permission. For an example policy and the key requirements, see Encrypting spans with a customer managed key.

Note

To use Transaction Search and other CloudWatch features, add the CloudWatchReadOnlyAccess policy to your role. For information about how to create a role, see IAM role creation.

Encrypting spans with a customer managed key

By default, CloudWatch Logs encrypts the `aws/spans` log group that stores your spans. To use your own key instead, choose a customer managed AWS KMS key when you enable Transaction Search. X-Ray applies the key only when it creates the `aws/spans` log group.

The key must meet the following requirements:

- It's a symmetric encryption key in the same account and Region as the log group. Specify the key ARN. Key IDs and alias ARNs aren't supported.
- The key policy allows CloudWatch Logs to use the key. For more information about key policies for CloudWatch Logs, see Encrypt log data in CloudWatch Logs using AWS KMS.
- The role that enables Transaction Search has the `kms:DescribeKey` permission for the key. If the `aws/spans` log group already exists, you also need the `logs:AssociateKmsKey` permission for it.

The following example JSON policy grants the `kms:DescribeKey` permission for one key. Add it to the role that enables Transaction Search:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TransactionSearchKmsDescribeKey",
            "Effect": "Allow",
            "Action": "kms:DescribeKey",
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/a1b2c3d4-5678-90ab-
cdef-EXAMPLE11111"
        }
    ]
}
```

In the example, make the following replacements:

- Replace *us-east-1* with the Region of your key.
- Replace *111122223333* with your account ID.
- Replace *a1b2c3d4-5678-90ab-cdef-EXAMPLE11111* with the ID of your key.

If the `aws/spans` log group already exists, X-Ray doesn't apply the key. In the console, CloudWatch applies your key to the existing log group in CloudWatch Logs instead.

If you use the API or CloudFormation, a request that includes a key fails, and X-Ray makes no changes. Instead, enable Transaction Search without a key, and then associate your key with the existing log group in CloudWatch Logs. Your key encrypts only spans that CloudWatch Logs ingests after you associate it.

The key applies only to the aws/spans log group. To encrypt the /aws/application-signals/data log group, create it with your key before you enable Transaction Search.

Enabling Transaction Search in the console

The following procedure describes how to enable Transaction Search in the console.

To enable Transaction Search in the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. From the navigation pane, under **Application Signals**, choose **Transaction Search**.
3. Choose **Enable Transaction Search**.
4. Select the box to ingest spans as structured logs, and enter a percentage of spans to be indexed. You can index spans at 1% for free and change the percentage later based on your requirements.
5. (Optional) Select **Customize encryption settings (advanced)**, and then choose a customer managed key to encrypt the aws/spans log group with your own key. For more information about the key requirements, see Encrypting spans with a customer managed key.

Enabling Transaction Search using an API

The following procedure describes how to enable Transaction Search using an API.

Step 1. Create a policy that grants access to ingest spans in CloudWatch Logs

When using the AWS CLI or SDK to enable Transaction Search, you must configure permissions using a resource-based policy with PutResourcePolicy.

Example policy

The following example policy allows X-Ray to send traces to CloudWatch Logs

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "TransactionSearchXRayAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "xray.amazonaws.com"
            },
            "Action": "logs:PutLogEvents",
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:aws/spans:*",
                "arn:aws:logs:us-east-1:123456789012:log-group:/aws/application-
signals/data:*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:xray:us-east-1:123456789012:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

Example command

The following example shows how to format your AWS CLI command with `PutResourcePolicy`.

```
aws logs put-resource-policy --policy-name MyResourcePolicy --policy-
document '{ "Version": "2012-10-17",        "Statement": [ { "Sid":
 "TransactionSearchXRayAccess", "Effect": "Allow", "Principal": { "Service":
 "xray.amazonaws.com" }, "Action": "logs:PutLogEvents", "Resource":
 [ "arn:partition:logs:region:account-id:log-group:aws/spans:*",
 "arn:partition:logs:region:account-id:log-group:/aws/application-signals/data:*" ],
 "Condition": { "ArnLike": { "aws:SourceArn": "arn:partition:xray:region:account-
id:*" }, "StringEquals": { "aws:SourceAccount": "account-id" } } } ]}'
```

Step 2. Configure the destination of trace segments

Configure the ingestion of spans with [UpdateTraceSegmentDestination](#).

Example command

The following example shows how to format your AWS CLI command with `UpdateTraceSegmentDestination`.

```
aws xray update-trace-segment-destination --destination CloudWatchLogs
```

To encrypt the `aws/spans` log group with your own key, add the key ARN in `CloudWatchLogsConfiguration`. For more information about the key requirements, see Encrypting spans with a customer managed key. In the following example, replace the key ARN with the ARN of your key.

```
aws xray update-trace-segment-destination --destination CloudWatchLogs --cloud-watch-
logs-configuration '{"SpansLogGroupKmsKeyId": "arn:partition:kms:region:account-
id:key/key-id"}'
```

Step 3. Configure the amount of spans to index

Configure your desired sampling percentage with `UpdateIndexingRule`

Example command

The following example shows how to format your AWS CLI command with `UpdateIndexingRule`.

```
aws xray update-indexing-rule --name "Default" --rule '{"Probabilistic":
 {"DesiredSamplingPercentage": number}}'
```

> **Note**
>
> After you enable Transaction Search, it can take 10 minutes for spans to become available for search and analysis.

Step 4. Verify spans are available for search and analysis

To verify spans are available for search and analysis, use `GetTraceSegmentDestination`.

Example commands

The following example shows how to format your AWS CLI command with `GetTraceSegmentDestination`.

```
aws xray get-trace-segment-destination
```

Example response

The following example shows the response you can expect when Transaction Search is active.

```
{
    "Destination": "CloudWatchLogs",
    "Status": "ACTIVE"
}
```

If you used a customer managed key, the following command shows which key encrypts the `aws/spans` log group.

```
aws logs describe-log-groups --log-group-name-prefix aws/spans --query
 'logGroups[].kmsKeyId'
```

Using Transaction Search with CloudFormation

You can use CloudFormation to enable and configure X-Ray Transaction Search.

> **Note**
>
> To create a CloudFormation stack, see Creating your first stack .

Prerequisites

- You must have access to an AWS account with an IAM user or role that has permissions to use Amazon EC2, Amazon S3, CloudFormation, or have administrative user access.
- You must have a Virtual Private Cloud (VPC) that has access to the internet. To keep things simple, you can use the default VPC that comes with your account. The default VPC and default subnets are sufficient for this configuration.
- Make sure Transaction Search is disabled before you enable using AWS CDK or CloudFormation.

Enabling Transaction Search

To enable Transaction Search using CloudFormation, you need to create the following two resources.

- `AWS::Logs::ResourcePolicy`
- `AWS::XRay::TransactionSearchConfig`

1. **Create AWS::Logs::ResourcePolicy** – Create a resource policy that allows X-Ray to send traces to CloudWatch Logs

   YAML

```
Resources:
  LogsResourcePolicy:
    Type: AWS::Logs::ResourcePolicy
    Properties:
      PolicyName: TransactionSearchAccess
      PolicyDocument: !Sub >
        {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "TransactionSearchXRayAccess",
              "Effect": "Allow",
              "Principal": {
                "Service": "xray.amazonaws.com"
              },
              "Action": "logs:PutLogEvents",
              "Resource": [
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:aws/spans:*",
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/application-signals/data:*"
              ],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": "arn:${AWS::Partition}:xray:${AWS::Region}:
${AWS::AccountId}:*"
                },
                "StringEquals": {
                  "aws:SourceAccount": "${AWS::AccountId}"
                }
              }
            }
          ]
```

```
        }
```

JSON

```
{
    "Resources": {
        "LogsResourcePolicy": {
            "Type": "AWS::Logs::ResourcePolicy",
            "Properties": {
                "PolicyName": "TransactionSearchAccess",
                "PolicyDocument": {
                    "Fn::Sub": "{\n  \"Version\": \"2012-10-17\",       \n
 \"Statement\": [\n    {\n      \"Sid\": \"TransactionSearchXRayAccess\",\n
     \"Effect\": \"Allow\",\n      \"Principal\": {\n        \"Service\":
 \"xray.amazonaws.com\"\n      },\n      \"Action\": \"logs:PutLogEvents\",\n
     \"Resource\": [\n        \"arn:${AWS::Partition}:logs:${AWS::Region}:
${AWS::AccountId}:log-group:aws/spans:*\",\n        \"arn:${AWS::Partition}:logs:
${AWS::Region}:${AWS::AccountId}:log-group:/aws/application-signals/data:*\"\n
  ],\n      \"Condition\": {\n        \"ArnLike\": {\n          \"aws:SourceArn\":
 \"arn:${AWS::Partition}:xray:${AWS::Region}:${AWS::AccountId}:*\"\n        },\n
     \"StringEquals\": {\n          \"aws:SourceAccount\": \"${AWS::AccountId}\"\n
       }\n      }\n    }\n  ]\n}"
                }
            }
        }
    }
}
```

2. **Create and Configure AWS::XRay::TransactionSearchConfig** – Create the `TransactionSearchConfig` resource to enable Transaction Search.

YAML

```
Resources:
  XRayTransactionSearchConfig:
    Type: AWS::XRay::TransactionSearchConfig
```

JSON

```
{
  "Resources": {
    "XRayTransactionSearchConfig": {
```

```
      "Type": "AWS::XRay::TransactionSearchConfig"
    }
  }
}
```

3. (Optional) You can set the `IndexingPercentage` property to control the percentage of spans that will be indexed.

   YAML

   ```
   Resources:
     XRayTransactionSearchConfig:
       Type: AWS::XRay::TransactionSearchConfig
       Properties:
         IndexingPercentage: 50
   ```

   JSON

   ```
   {
     "Resources": {
       "XRayTransactionSearchConfig": {
         "Type": "AWS::XRay::TransactionSearchConfig",
         "Properties": {
           "IndexingPercentage": 20
         }
       }
     }
   }
   ```

   The IndexingPercentage value can be set between 0 and 100.

4. (Optional) To encrypt the `aws/spans` log group with your own key, set the `CloudWatchLogsConfiguration` property. For `SpansLogGroupKmsKeyId`, specify the key ARN. If the key is an `AWS::KMS::Key` resource in the same template, use `Fn::GetAtt` with `Arn`. Don't use `Ref`, which returns the key ID. For more information about the key requirements, see Encrypting spans with a customer managed key.

   YAML

   ```
   Resources:
     XRayTransactionSearchConfig:
       Type: AWS::XRay::TransactionSearchConfig
   ```

```
    Properties:
      CloudWatchLogsConfiguration:
        SpansLogGroupKmsKeyId: !GetAtt SpansLogGroupKey.Arn
```

JSON

```
{
  "Resources": {
    "XRayTransactionSearchConfig": {
      "Type": "AWS::XRay::TransactionSearchConfig",
      "Properties": {
        "CloudWatchLogsConfiguration": {
          "SpansLogGroupKmsKeyId": { "Fn::GetAtt": ["SpansLogGroupKey", "Arn"] }
        }
      }
    }
  }
}
```

X-Ray uses the key only when this resource enables Transaction Search and creates the `aws/spans` log group. If the log group already exists, CloudFormation fails to create the resource. Adding, changing, or removing `CloudWatchLogsConfiguration` replaces the resource, which fails while Transaction Search is enabled. To change the key later, use the CloudWatch Logs AssociateKmsKey operation.

Deleting the stack turns off Transaction Search but keeps the `aws/spans` log group. A later stack that sets `SpansLogGroupKmsKeyId` then fails to create.

Template examples

The following example includes both the resource policy and the TransactionSearchConfig.

YAML

```
Resources:
  LogsResourcePolicy:
    Type: AWS::Logs::ResourcePolicy
    Properties:
      PolicyName: TransactionSearchAccess
      PolicyDocument: !Sub >
        {
```

```
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "TransactionSearchXRayAccess",
              "Effect": "Allow",
              "Principal": {
                "Service": "xray.amazonaws.com"
              },
              "Action": "logs:PutLogEvents",
              "Resource": [
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:aws/spans:*",
                "arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/application-signals/data:*"
              ],
              "Condition": {
                "ArnLike": {
                  "aws:SourceArn": "arn:${AWS::Partition}:xray:${AWS::Region}:
${AWS::AccountId}:*"
                },
                "StringEquals": {
                  "aws:SourceAccount": "${AWS::AccountId}"
                }
              }
            }
          ]
        }

  XRayTransactionSearchConfig:
    Type: AWS::XRay::TransactionSearchConfig
    Properties:
      IndexingPercentage: 10
```

JSON

```
{
    "Resources": {
        "LogsResourcePolicy": {
            "Type": "AWS::Logs::ResourcePolicy",
            "Properties": {
                "PolicyName": "TransactionSearchAccess",
                "PolicyDocument": {
```

```
                    "Fn::Sub": "{\n  \"Version\": \"2012-10-17\",       \n  \"Statement
\": [\n    {\n      \"Sid\": \"TransactionSearchXRayAccess\",\n      \"Effect\":
 \"Allow\",\n      \"Principal\": {\n        \"Service\": \"xray.amazonaws.com\"\n
      },\n      \"Action\": \"logs:PutLogEvents\",\n      \"Resource\": [\n
 \"arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-group:aws/spans:*
\",\n        \"arn:${AWS::Partition}:logs:${AWS::Region}:${AWS::AccountId}:log-
group:/aws/application-signals/data:*\"\n      ],\n      \"Condition\": {\n
     \"ArnLike\": {\n          \"aws:SourceArn\": \"arn:${AWS::Partition}:xray:
${AWS::Region}:${AWS::AccountId}:*\"\n        },\n        \"StringEquals\": {\n
  \"aws:SourceAccount\": \"${AWS::AccountId}\"\n        }\n      }\n    }\n  ]\n}"
                }
            }
        },
        "XRayTransactionSearchConfig": {
            "Type": "AWS::XRay::TransactionSearchConfig",
            "Properties": {
                "IndexingPercentage": 20
            }
        }
    }
}
```

Here is an example using AWS CDK in TypeScript.

CDK

```
import * as cdk from '@aws-cdk/core';
import * as logs from '@aws-cdk/aws-logs';
import * as xray from '@aws-cdk/aws-xray';

export class XRayTransactionSearchStack extends cdk.Stack {
  constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the resource policy
    const transactionSearchAccess = new logs.CfnResourcePolicy(this,
 'XRayLogResourcePolicy', {
      policyName: 'TransactionSearchAccess',
      policyDocument: JSON.stringify({
        Version: '2012-10-17',
        Statement: [
          {
            Sid: 'TransactionSearchXRayAccess',
            Effect: 'Allow',
```

```
            Principal: {
              Service: 'xray.amazonaws.com',
            },
            Action: 'logs:PutLogEvents',
            Resource: [
              `arn:${this.partition}:logs:${this.region}:${this.account}:log-group:aws/
spans:*`,
              `arn:${this.partition}:logs:${this.region}:${this.account}:log-group:/
aws/application-signals/data:*`,
            ],
            Condition: {
              ArnLike: {
                'aws:SourceArn': `arn:${this.partition}:xray:${this.region}:
${this.account}:*`,
              },
              StringEquals: {
                'aws:SourceAccount': this.account,
              },
            },
          },
        ],
      }),
    });

    // Create the TransactionSearchConfig with dependency
    const transactionSearchConfig = new xray.CfnTransactionSearchConfig(this,
 'XRayTransactionSearchConfig', {
      indexingPercentage: 10,
    });

    // Add the dependency to ensure Resource Policy is created first
    transactionSearchConfig.addDependsOn(transactionSearchAccess);
  }
}
```

Verifying the configuration

After deploying your CloudFormation stack, you can verify the configuration using the AWS CLI.

aws xray get-trace-segment-destination

A successful configuration will return the following.

```
{
    "Destination": "CloudWatchLogs",
    "Status": "ACTIVE"
}
```

Spans

Spans sent to X-Ray are ingested and managed in a log group called `aws/spans`. This topic describes which CloudWatch Logs features are available for transaction spans.

By default, CloudWatch Logs encrypts the `aws/spans` log group. To encrypt it with a customer managed AWS KMS key instead, see Encrypting spans with a customer managed key.

Available features

The following CloudWatch Logs features are available for transaction spans.

- Metric filters – Use metric filters to extract custom metrics from spans.
- Subscriptions – Use subscriptions to access a real-time feed of span events from CloudWatch Logs.
- Log anomaly detection – Use log anomaly detection to establish a baseline for spans sent to the `aws/spans` log group.
- Contributor Insights – Use Contributor Insights to analyze span data and create a time series displaying contributor data.

Unsupported features

The following are features not supported for transaction spans.

- Spans cannot be sent to CloudWatch Logs with the `PutLogEvents` API.
- Span data cannot be enriched or transformed.

> **Note**
>
> Span ingestion is charged separately from log ingestion. For information about pricing, see Amazon CloudWatch Pricing.

Searching and analyzing spans

Transaction Search provides you with a visual editor to search and analyze all ingested spans using attributes. You can use the visual editor to narrow down transaction spans and create interactive visualizations to troubleshoot issues in your distributed applications. You can also use the CloudWatch Logs Insights query language to analyze your spans. This topic describes how to access and use the visual editor.

The visual editor

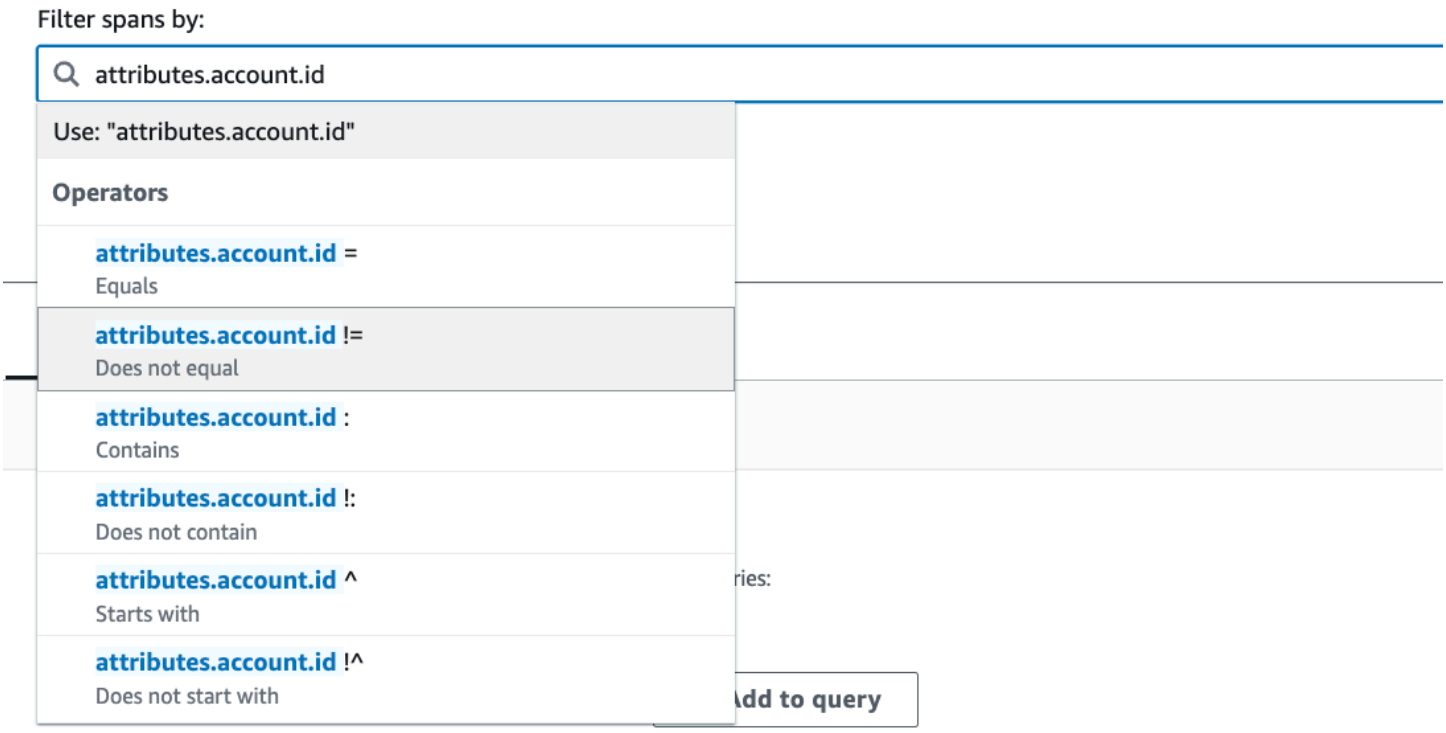
The following procedure describes how to access the visual editor.

To access the visual editor

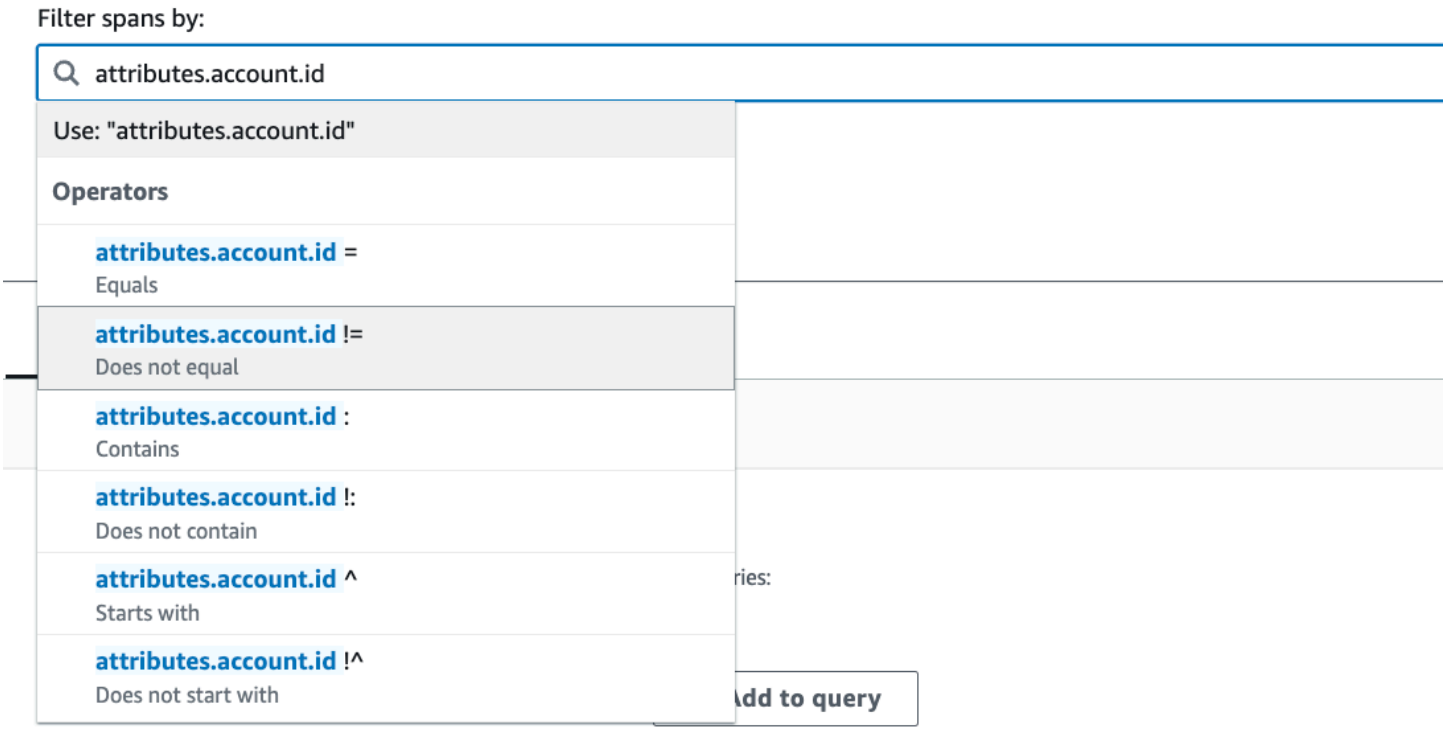
1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. From the navigation pane, choose **Application Signals**, and then choose **Transaction Search**.

Use span attributes, such as service name, span duration, and span status to narrow down transaction spans quickly. You can access these filters and more on the right side of the visual editor under **Select filters**.

This visual editor suggests a list of attributes in the span. These attributes include attributes added through auto-instrumentation and custom attributes added through custom instrumentation.



Select a span key, and enter a value to refine span results. You can filter spans using various operations, such as "Equals," "Does Not Equal," and more.



Query formats

You can run queries in the visual editor using different formats. This section describes each of these formats.

List

View spans or span events in a list format, which displays information about each span. Use this type of analysis to analyze individual spans, understand specific transactions, or identify unique patterns in transaction events. Other use cases include the following:

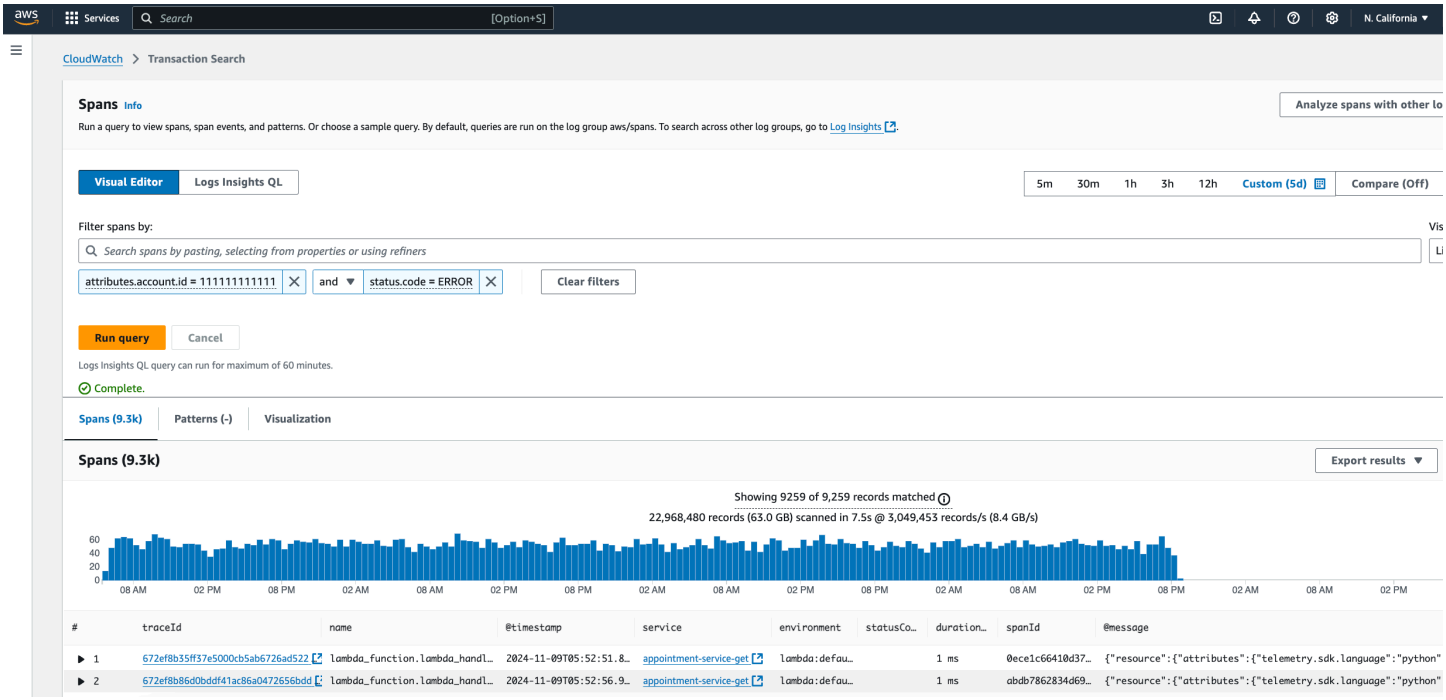
Use cases

- Troubleshoot customer support tickets
- Locate APIs or dependencies, such as database queries taking longer than 1000 milliseconds to execute
- Locate spans with errors

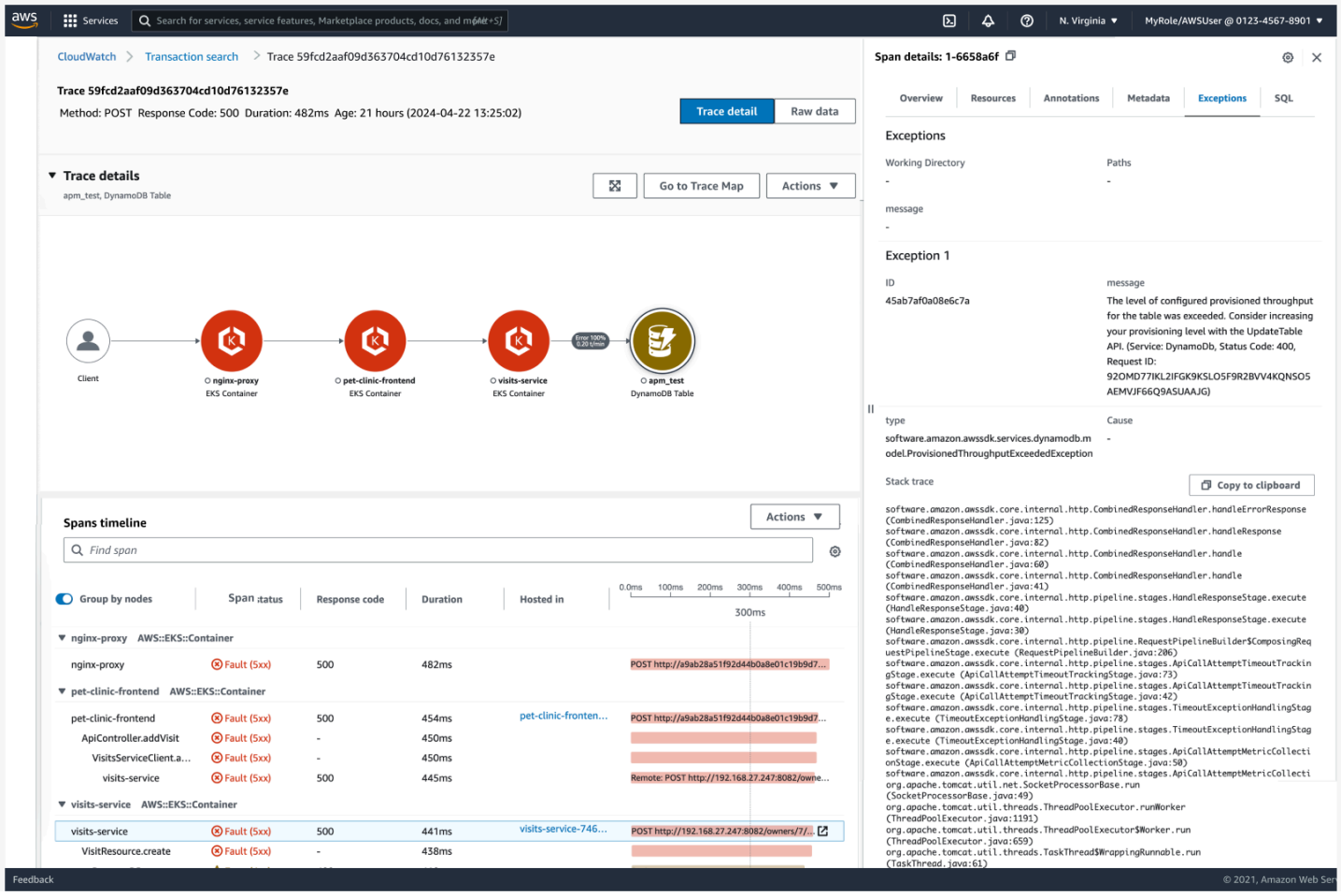
The following screenshots show how to troubleshoot a customer support ticket with this type of analysis.

Example scenario

In the visual editor, filter on all transaction spans with a particular customer issue. Before you run your query, choose **List** from the **Visualize as** dropdown.



The results show a list of spans where you can choose a trace ID to get the end-to-end journey for the transaction and determine the root cause of the issue.



Timeseries

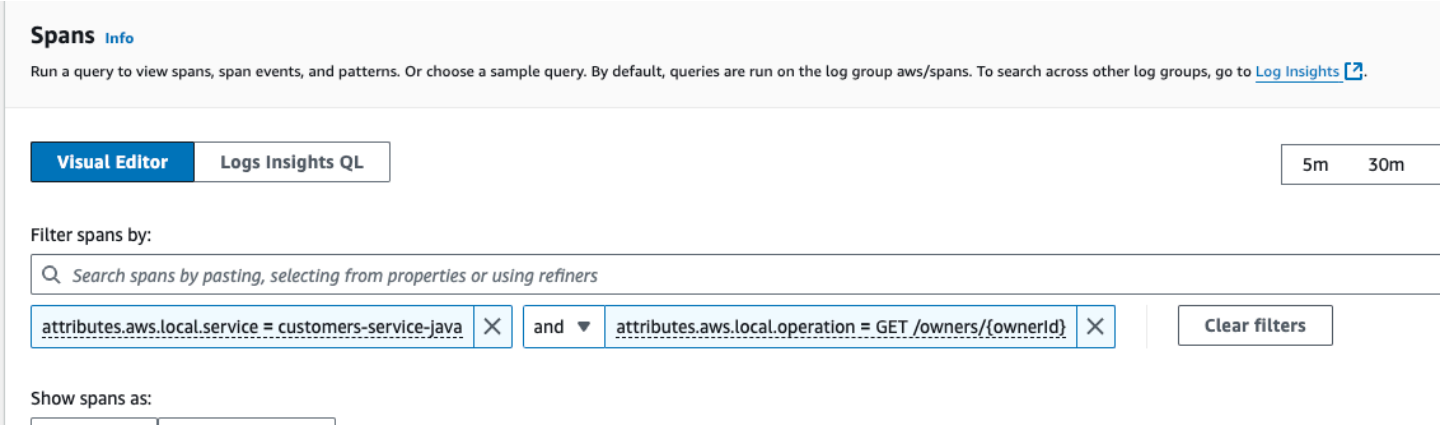
View spans or span events over time. Use this type of analysis to look at trends and spikes in transaction activity. Other use cases include the following:

- Visualize latency
- Visualize frequency of spans
- Visualize performance

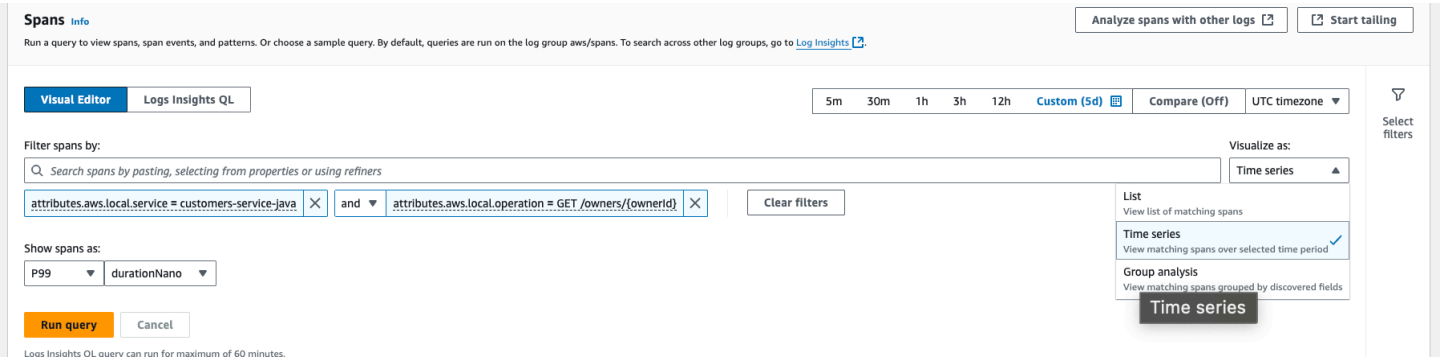
The following screenshots show how you can view p99 latency trends for an API with this type of analysis.

Example scenario

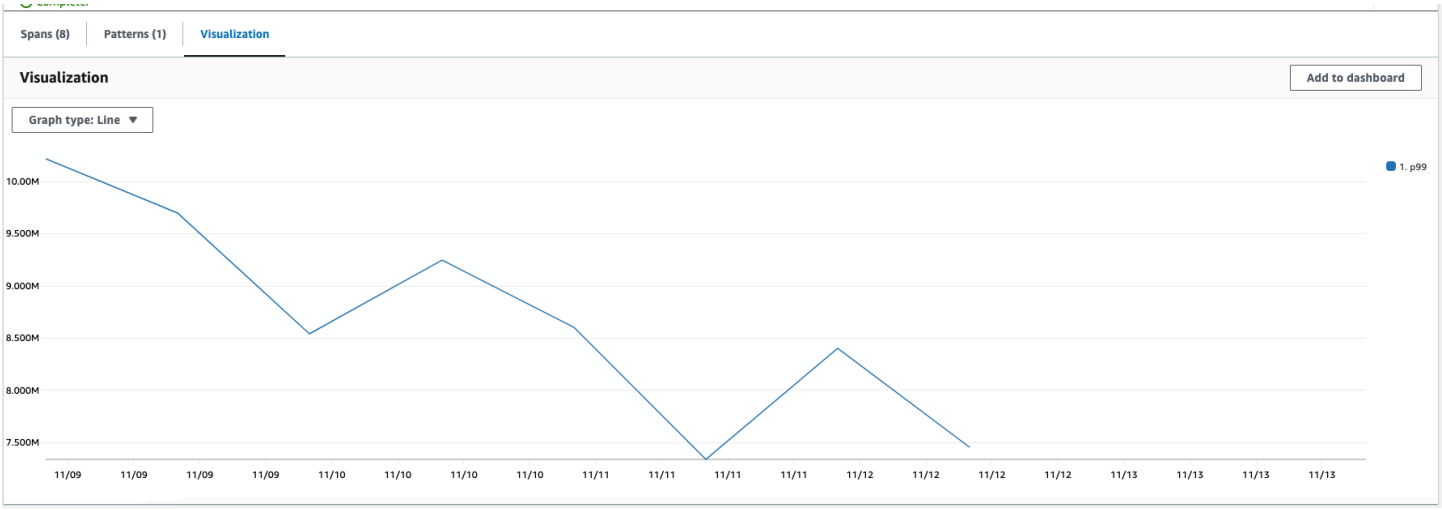
In the visual editor, filter on the service and API you want to analyze.



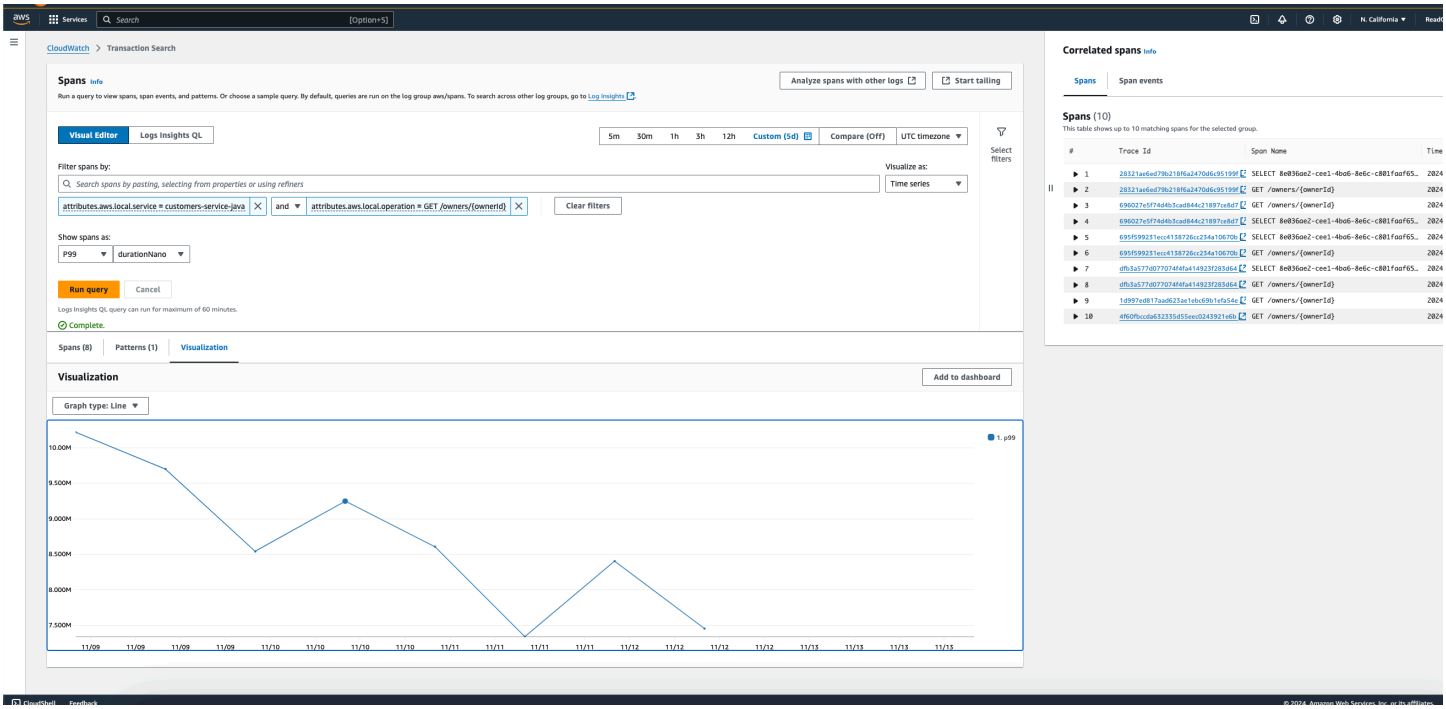
Before you run your query, choose **Time series** from the **Visualize as** dropdown. Choose **P99** for the duration statistic from the **Show span as** dropdown.



The results show a latency trend for the service, with the x-axis of the graph being time and y-axis being p99 duration.



You can choose a point on the chart to view correlated spans and span events.



Group analysis

Aggregate spans or span events based on specific attributes, such as account IDs and status codes, to display statistical metrics. Use this type of analysis to analyze spans in clusters, compare different groups, and uncover trends at the macro level. Other use cases include the following:

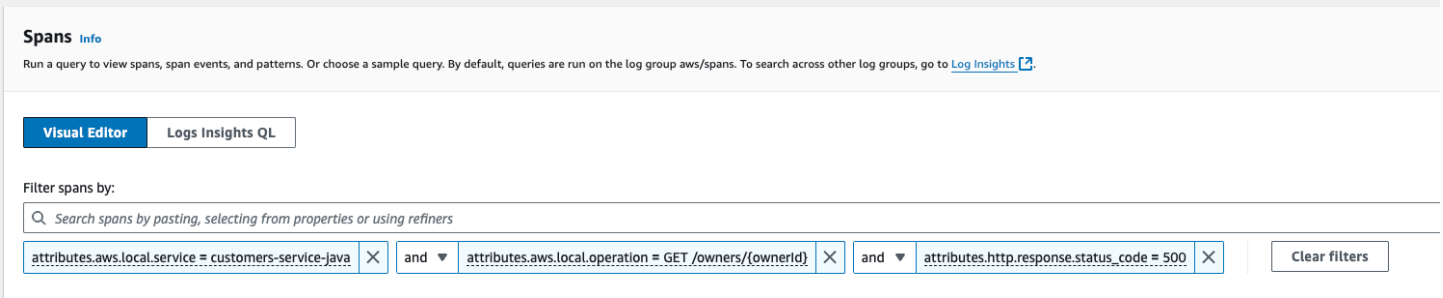
Use cases

- Identify top customers impacted by a service outage
- Identify availability zones with the most errors
- Identify the top slowest database queries

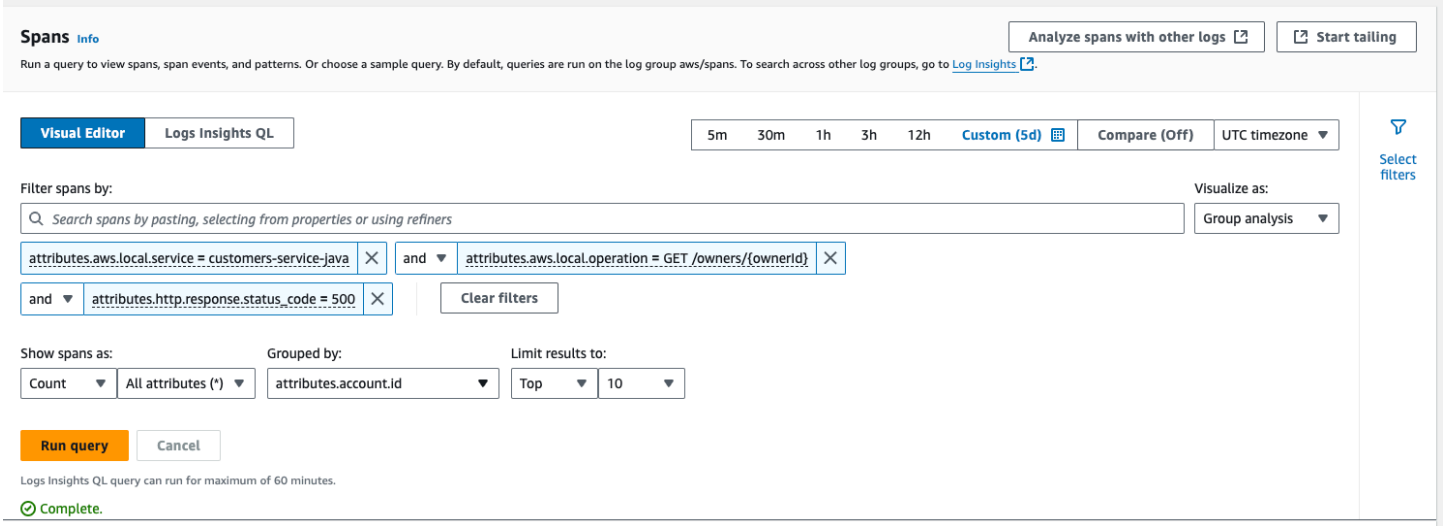
The following screenshots show how you can view the top customers impacted by a service outage with this type of analysis.

Example scenario

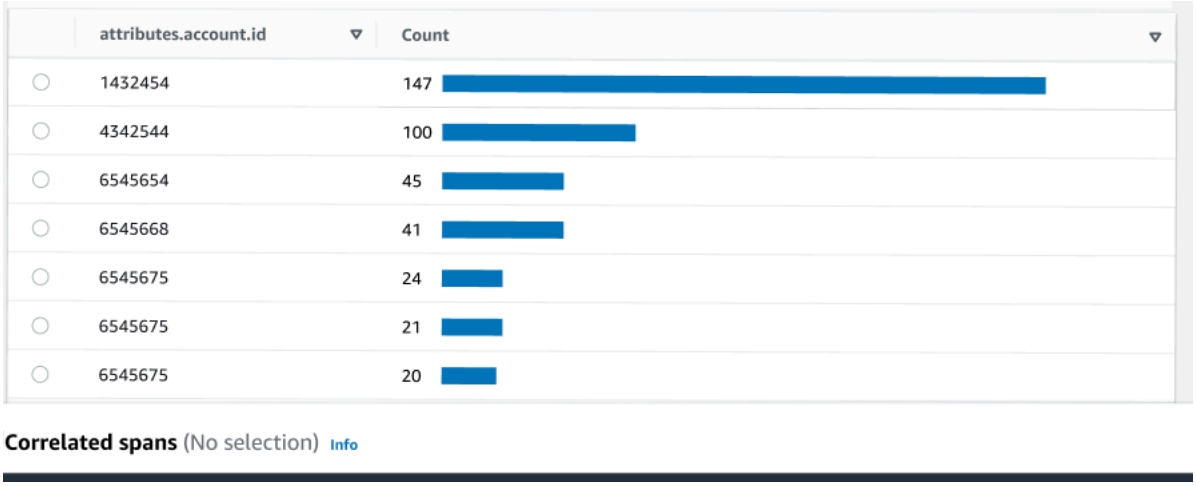
In the visual editor, you filter on the service experiencing issues.



Before you run your query, choose **Group Analysis** from the **Visualize as** dropdown. Group your query results by `account.id`, and limit the number of results to 10..



The results show the top 10 customers who experienced the most number of errors.



CloudWatch Logs Insights

You can use CloudWatch Logs Insights to analyze your spans.

Example query

The following query shows the top five slowest database queries.

```
STATS pct(durationNano, 99) as `p99` by attributes.db.statement
| SORT p99 ASC
| LIMIT 5
| DISPLAY p99,attributes.db.statement
```

Example query

The following query shows which top five services are throwing errors.

```
FILTER `attributes.http.response.status_code` >= 500
| STATS count(*) as `count` by attributes.aws.local.service as service
| SORT count ASC
| LIMIT 5
| DISPLAY count,service
```

Ingesting spans for complete visibility

Recording all transaction spans provides comprehensive visibility into application issues. It enables you to debug customer support tickets or troubleshoot rarely occurring p99 API latency spikes, which is crucial when identifying the root cause of issues in customer-facing and mission-critical applications. You can create a cost-effective strategy to start capturing 100% of trace spans in CloudWatch by configuring the head sampling rate and then adjusting a lower span indexing rate.

Setting up head sampling

Head sampling is a tracing technique capturing requests at the beginning of a trace, which is based on a set rate or condition.

When the head sampling rate is set to 100%, it captures the beginning of every trace without skips, guaranteeing complete visibility into all incoming requests, and that no transaction data is missed.

You can configure head sampling if you're using X-Ray or AWS Distro for OpenTelemetry SDKs or the OpenTelemetry SDK.

If you're using X-Ray or AWS Distro for OpenTelemetry SDKs

Navigate to your sampling rules in the console, and set the fixed sampling rate to 100%. This guarantees all trace spans are captured and ingested into CloudWatch logs. For more information, see Configuring sampling rules

If you're using the OpenTelemetry SDK

To record 100% of spans and get complete visibility, set your sampling configuration to always_on. For more information, see Language APIs & SDKs on the OpenTelemetry website.

Features unlocked with head sampling

When you enable Transaction Search, all spans collected from your application through head sampling are ingested as structured logs in CloudWatch. This provides you with the following features:

- The ability to search span attributes and analyze span events in a visual editor.
- The ability to visualize traces containing up to 10,000 spans.
- Total support for OpenTelemetry, which includes the ability to embed business events into spans for analysis and use span links to define connections between traces for end-to-end viewing.
- Access to application dashboards, metrics, and topology with CloudWatch Application Signals enabled for all spans sent to CloudWatch.

> **Note**
>
> Because spans are available in a log group called `aws/spans`, you can use CloudWatch Logs features with transaction spans. For more information, see The span log group.

Setting up span indexing with trace summaries

Trace summaries can help you debug transactions and are beneficial for asynchronous processes. You only need to index a small percentage of spans as trace summaries.

You configure span indexing when you enable Transaction Search in the console or with the API. To enable Transaction Search, see Getting started with Transaction Search.

Features unlocked with trace summaries

The key features of X-Ray trace summaries include the following:

- Trace summary search – Search and find traces from trace summaries.
- Trace summary analytics – Interpret trace data.
- Trace insights – Analyze trace data to identify application issues.

Monitoring spans across accounts

Spans sent to X-Ray are ingested and managed in a log group called `aws/spans`. To monitor spans across multiple accounts, you must enable Transaction Search across all source and monitoring accounts and enable cross-account observability for logs and traces. When you enable cross-account observability, you can search up to 10,000 accounts and get visibility into traces across accounts. This feature is provided at no extra cost for the `aws/spans` log group. If you enable cross-account observability for trace summaries, the first trace summary copy is free. For more information about pricing, see Amazon CloudWatch Pricing.

Adding custom attributes

CloudWatch Application Signals uses OpenTelemetry to auto-instrument your applications and collect spans from popular libraries in different languages, such as Java, Python, and more.

Auto-instrumentation captures information, such as database queries, HTTP requests, cache accesses, and external service calls, which allows you to troubleshoot application performance issues.

You can add custom instrumentation to enrich spans with business-specific data or other information you wish to capture. This data can be recorded as a custom attribute or a span event, providing insights tailored to your troubleshooting needs.

> **Note**
>
> For information about adding custom attributes or span events in a different language, see Language APIs and SDKS in the *OpenTelemetry website*.

Custom attributes

You can add business related attributes or any other attributes to your spans in all languages OpenTelemetry supports. The following is a Java code snippet that adds an order id and customer details to a span.

```
import io.opentelemetry.api.trace.Span;

public class OrderProcessor {

    public void processOrder() {
        Span span = Span.current();
        span.setAttribute("order.id", "123456");
        span.setAttribute("customer.name", "John Doe");
        span.setAttribute("customer.id", "4343dfdd");

        // Your order processing logic here
        System.out.println("Order processed with custom attributes");
    }
}
```

After these attributes have been added to the span, they become available to search and analyze in the Transaction Search visual editor.

Span events

A span event is typically used to denote a meaningful, singular point in time during a span duration. Exceptions are auto-captured as span events through auto-instrumentation, but you can also add custom business events, such as payment-status or cart-abandonment. For more information, see Span events on the OpenTelemetry website.

You can embed span events to your spans in all the languages CloudWatch Application Signals and OpenTelemetry support. The following is a Java code snippet of adding a custom event to a span.

```
import io.opentelemetry.api.trace.Span;

public class OrderProcessor {

    public void bookOrder() {
        Span span = Span.current();
```

```
        // Add a booking started event
        span.addEvent("booking started");

        // Add a payment succeeded event or failed event
        span.addEvent("booking failed");
    }
}
```

Prerequisites for the CloudWatch agent

When using the CloudWatch agent to emit span events to X-Ray, you must turn on the `transit_spans_in_otlp_format` flag in your configuration.

```
{
  "traces": {
    ...
    "transit_spans_in_otlp_format": true
    ...
  }
}
```

After you add these events, they become available in the Transaction Search visual editor.

CloudWatch Logs queries

You can query span events in CloudWatch Logs to view advanced insights. The following example query commands show how to analyze exceptions thrown by your application:

```
fields jsonparse(@message) as js
| unnest js.events into event
| filter event.name = "exception"
| display event.attributes.`exception.stacktrace`
```

```
fields jsonparse(@message) as js
| unnest js.events into event
| filter event.name = "exception"
| stats count() by event.attributes.`exception.type`
```

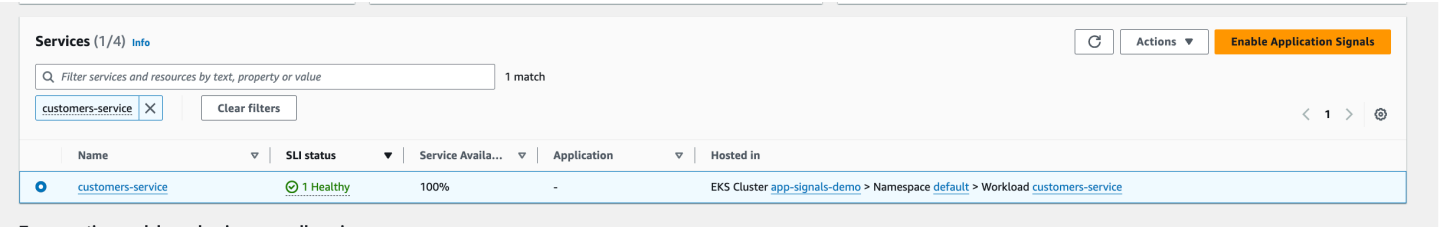
Troubleshooting application issues

With Application Signals, you can troubleshoot rarely occurring latency spikes in your applications. After you enable Transaction Search and configure a head sampling rate capturing 100% of spans, you get complete visibility into any application issue. The following scenario describes how Application Signals can be used with transaction spans to monitor your services and identify service quality issues.

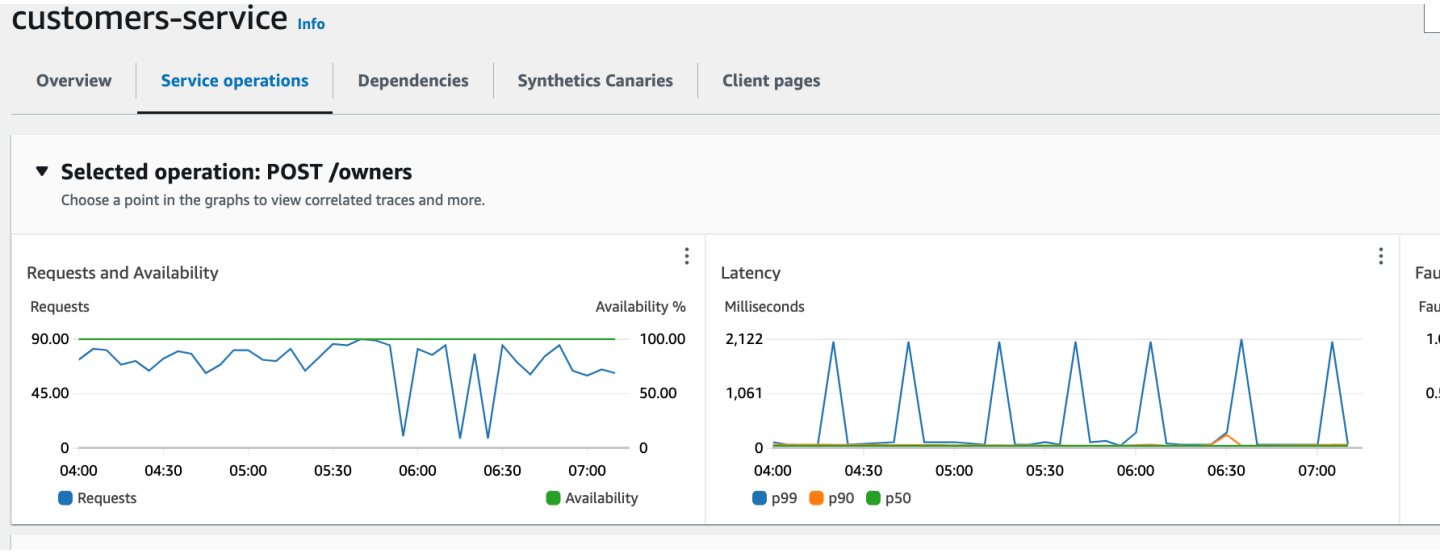
Example troubleshooting scenario

This scenario focuses on a pet clinic application composed of several micro-services calling third-party payment APIs. These calls have been intermittently slow, thus impacting revenue.

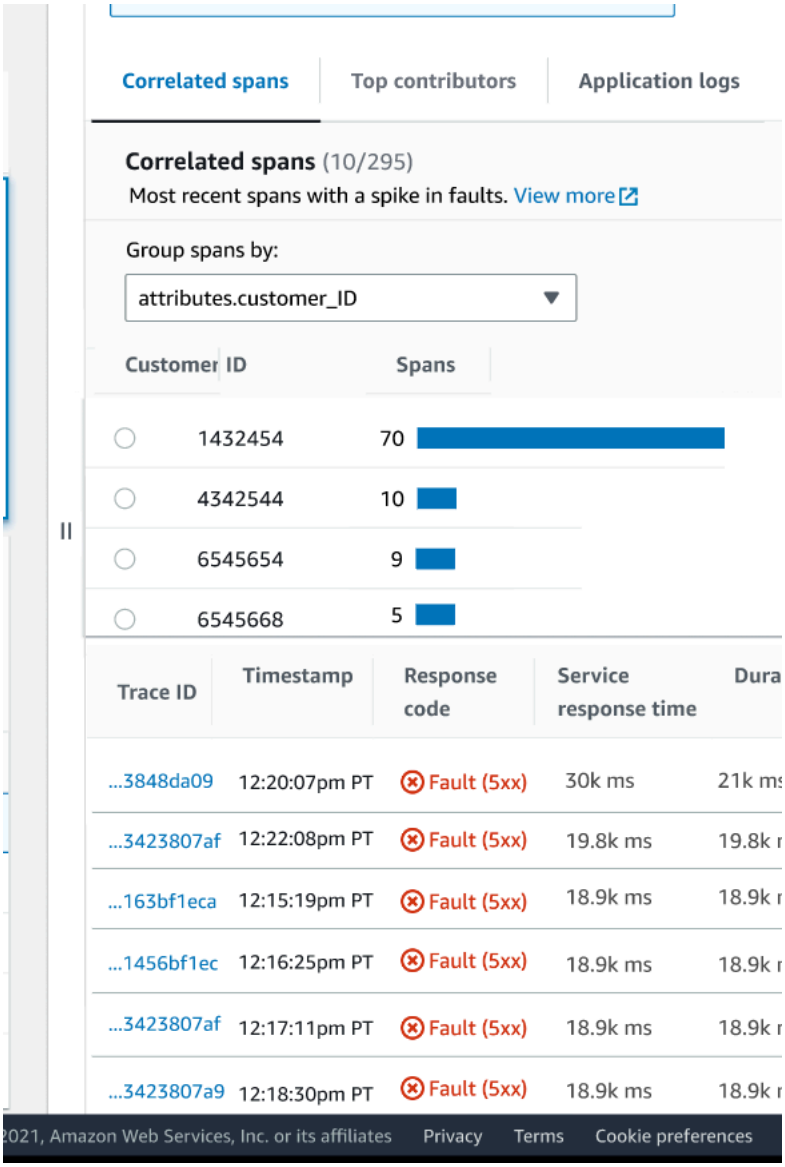
Jane opens the CloudWatch Application Signals console and notices a customer-service application responsible for registering customers is healthy and not breaching any SLOs.



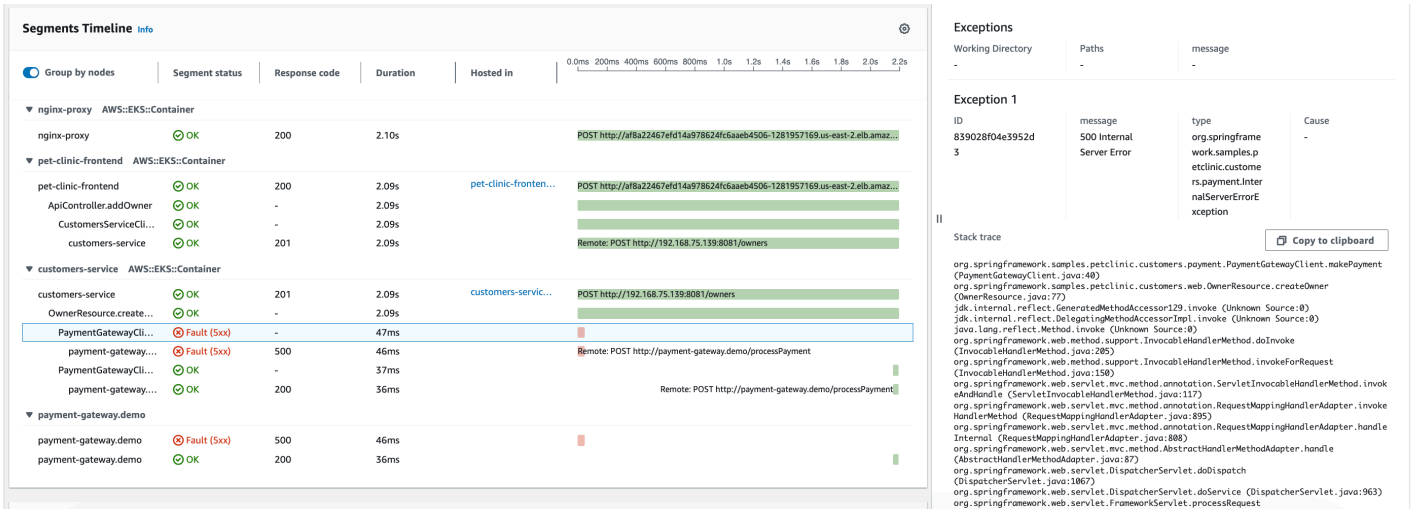
She opens the service to investigate any patterns of rarely occurring failures and notices the registration API experienced intermittent p99 latency spikes.



Jane chooses a datapoint in the latency chart to view correlated spans. She groups the spans by customer ID to view all the customers who are impacted by the latency spikes.



Jane selects one of the correlated spans with a fault status, which opens the trace detail page for the selected trace. She scrolls to the segments timeline section to follow the call path, where she notices that calls to the payment gateway have been failing and preventing customers from registering.



Synthetic monitoring (canaries)

You can use Amazon CloudWatch Synthetics to create *canaries*, configurable scripts that run on a schedule, to monitor your endpoints and APIs. Canaries follow the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your applications. By using canaries, you can discover issues before your customers do.

Canaries are scripts written in Node.js, Python, or Java. They create Lambda functions in your account that use Node.js, Python, or Java as the runtime. Canaries work over both HTTP and HTTPS protocols. Canaries use Lambda layers that contain the CloudWatch Synthetics library. The library includes CloudWatch Synthetics implementations for NodeJS, Python, and Java.

Canaries in Node.js and Python runtimes offer programmatic access to headless browsers through Playwright, Puppeteer, or Selenium Webdriver. Multiple browsers are supported, including a headless Google Chrome browser, and Mozilla Firefox. For more information about Playwright, see Playwright. For more information about Puppeteer, see Puppeteer. For more information about Selenium, see Selenium. Canaries on Selenium only support Chrome browser. Canaries in Java are designed for flexibility in monitoring any type of service or application and do not contain browser support or frameworks.

Canaries check the availability and latency of your endpoints and can store load time data and screenshots of the UI. They monitor your REST APIs, URLs, and website content, and they can check for unauthorized changes from phishing, code injection and cross-site scripting.

CloudWatch Synthetics is integrated with Application Signals, which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see your canaries in Application Signals, turn on X-Ray active tracing. Your canaries are displayed on the Application Map connected to your services, and in the Service detail page of the services they call.

For a video demonstration of canaries, see the following:

- Introduction to Amazon CloudWatch Synthetics
- Amazon CloudWatch Synthetics Demo
- Create Canaries Using Amazon CloudWatch Synthetics
- Visual Monitoring with Amazon CloudWatch Synthetics

You can run a canary once or on a regular schedule. Canaries can run as often as once per minute. You can use both cron and rate expressions to schedule canaries.

For information about security issues to consider before you create and run canaries, see Security considerations for Synthetics canaries.

By default, canaries create several CloudWatch metrics in the `CloudWatchSynthetics` namespace. These metrics have `CanaryName` as a dimension. Canaries that use the `executeStep()` or `executeHttpStep()` function from the function library also have `StepName` as a dimension. For more information about the canary function library, see Library functions available for canary scripts.

CloudWatch Synthetics integrates well with the X-Ray Trace Map, which uses CloudWatch with AWS X-Ray to provide an end-to-end view of your services to help you more efficiently pinpoint performance bottlenecks and identify impacted users. Canaries that you create with CloudWatch Synthetics appear on the trace map. For more information, see X-Ray Trace Map.

CloudWatch Synthetics is currently available in all commercial AWS Regions and the GovCloud Regions.

Topics

- Required roles and permissions for CloudWatch canaries
- Creating a canary

- Multilocation canaries
- Groups
- Test a canary locally
- Troubleshooting a failed canary
- Sample code for canary scripts
- Canaries and X-Ray tracing
- Running a canary on a VPC
- Encrypting canary artifacts
- Viewing canary statistics and details
- CloudWatch metrics published by canaries
- Edit or delete a canary
- Start, stop, delete, or update runtime for multiple canaries
- Monitoring canary events with Amazon EventBridge
- Performing safe canary updates

Required roles and permissions for CloudWatch canaries

Both the users who create and manage canaries, and the canaries themselves, must have certain permissions.

Required roles and permissions for users who manage CloudWatch canaries

To view canary details and the results of canary runs, you must be signed in as a user with either the `CloudWatchSyntheticsFullAccess` or the `CloudWatchSyntheticsReadOnlyAccess` policies attached. To read all Synthetics data in the console, you also need the `AmazonS3ReadOnlyAccess` and `CloudWatchReadOnlyAccess` policies. To view the source code used by canaries, you also need the `AWSLambda_ReadOnlyAccess` policy.

To create canaries, you must be signed in as an user who has the `CloudWatchSyntheticsFullAccess` policy or a similar set of permissions. To create IAM roles for the canaries, you also need the following inline policy statement:

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*",
                "arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*"
            ]
        }
    ]
}
```

Important

Granting a user the `iam:CreateRole`, `iam:CreatePolicy`, and `iam:AttachRolePolicy` permissions gives that user full administrative access to your AWS account. For example, a user with these permissions can create a policy that has full permissions for all resources and can attach that policy to any role. Be very careful about who you grant these permissions to.

For information about attaching policies and granting permissions to users, see Changing Permissions for an IAM User and To embed an inline policy for a user or role.

Required roles and permissions for canaries

Each canary must be associated with an IAM role that has certain permissions attached. When you create a canary using the CloudWatch console, you can choose for CloudWatch Synthetics to create an IAM role for the canary. If you do, the role will have the permissions needed.

If you want to create the IAM role yourself, or create an IAM role that you can use when using the AWS CLI or APIs to create a canary, the role must contain the permissions listed in this section.

All IAM roles for canaries must include the following trust policy statement.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Additionally, the canary's IAM role needs one of the following statements.

Basic canary that doesn't use AWS KMS or need Amazon VPC access

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/s3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/s3/bucket/that/contains/canary/results"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        }
    ]
}
```

Canary that uses AWS KMS to encrypt canary artifacts but does not need Amazon VPC access

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.us-east-1.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Canary that does not use AWS KMS but does need Amazon VPC access

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Canary that uses AWS KMS to encrypt canary artifacts and also needs Amazon VPC access

If you update a non-VPC canary to start using a VPC, you'll need to update the canary's role to include the network interface permissions listed in the following policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::path/to/your/S3/bucket/canary/results/folder"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::name/of/the/S3/bucket/that/contains/canary/results"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup"
            ],
            "Resource": [
            "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
cwsyn-canary_name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "xray:PutTraceSegments"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CloudWatchSynthetics"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": [
                "*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/KMS_key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "s3.us-east-1.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Canary that uses a customer managed AWS KMS key to encrypt Lambda function environment variables

If you configure a customer managed key for environment variable encryption at rest, you must have `kms:CreateGrant`, `kms:Encrypt`, `kms:Decrypt`, and `kms:DescribeKey` permissions on the key. The canary execution role must also have `kms:Decrypt` if you use encryption in transit. For more information, see Encrypting environment variables at rest with a customer managed key.

If you use encryption in transit, add the following statement to your canary's execution role policy:

```
{
  "Effect": "Allow",
  "Action": "kms:Decrypt",
  "Resource": "arn:aws:kms:us-east-1:111122223333:key/a1b2c3d4-e5f6-7890-abcd-
EXAMPLE11111"
}
```

Add this statement to whichever base execution role policy you are already using (basic, VPC, or artifact KMS).

AWS managed policies for CloudWatch Synthetics

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that

provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies AWS managed policies in the IAM User Guide.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally change the permissions in an AWS managed policy. This type of update affects all identities (users, groups, and roles) where the policy is attached.

CloudWatch Synthetics updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch Synthetics since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
Redundant actions removed from **CloudWatchSyntheticsFullAccess**	CloudWatch Synthetics removed the `s3:PutBucketEncryption` and `lambda:GetLayerVersionByArn` actions from **CloudWatchSyntheticsFullAccess** policy because those actions were redundant with other permissions in the policy. The removed actions did not provide any permissions, and there's no net change to the permissions granted by the policy.	March 12, 2021
CloudWatch Synthetics started tracking changes	CloudWatch Synthetics started tracking changes for its AWS managed policies.	March 10, 2021

CloudWatchSyntheticsFullAccess

Here are the contents of the `CloudWatchSyntheticsFullAccess` policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "synthetics:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": [
                "arn:aws:s3:::cw-syn-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "xray:GetTraceSummaries",
                "xray:BatchGetTraces",
                "apigateway:GET"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::cw-syn-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::aws-synthetics-library-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "lambda.amazonaws.com",
                        "synthetics.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": [
                "arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
```

```
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DeleteAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws:cloudwatch:*:*:alarm:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetLogRecord",
                "logs:DescribeLogStreams",
                "logs:StartQuery",
                "logs:GetLogEvents",
                "logs:FilterLogEvents",
                "logs:GetLogGroupFields"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/lambda/cwsyn-*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": "${aws:PrincipalAccount}"
                }
            }
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:PublishVersion",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunction",
                "lambda:DeleteFunction",
                "lambda:ListTags",
                "lambda:TagResource",
                "lambda:UntagResource"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:function:cwsyn-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": [
                "arn:aws:lambda:*:*:layer:cwsyn-*",
                "arn:aws:lambda:*:*:layer:Synthetics:*",
                "arn:aws:lambda:*:*:layer:Synthetics_Selenium:*",
                "arn:aws:lambda:*:*:layer:AWS-CW-Synthetics*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "sns:ListTopics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:Subscribe",
                "sns:ListSubscriptionsByTopic"
            ],
            "Resource": [
                "arn:*:sns:*:*:Synthetics-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:*:*:key/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:*:*:key/*",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "s3.*.amazonaws.com"
                    ]
                }
```

```
            }
        }
    ]
}
```

CloudWatchSyntheticsReadOnlyAccess

Here are the contents of the `CloudWatchSyntheticsReadOnlyAccess` policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:Describe*",
                "synthetics:Get*",
                "synthetics:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Limiting a user to viewing specific canaries

You can limit a user's ability to view information about canaries, so that they can only see information about the canaries you specify. To do this, use an IAM policy with a `Condition` statement similar to the following, and attach this policy to a user or an IAM role.

The following example limits the user to only viewing information about `name-of-allowed-canary-1` and `name-of-allowed-canary-2`.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "synthetics:Names": [
                        "name-of-allowed-canary-1",
                        "name-of-allowed-canary-2"
                    ]
                }
            }
        }
    ]
}
```

CloudWatch Synthetics supports listing as many as five items in the `synthetics:Names` array.

You can also create a policy that uses a * as a wildcard in canary names that are to be allowed, as in the following example:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "synthetics:DescribeCanaries",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringLike": {
                    "synthetics:Names": [
                        "my-team-canary-*"
                    ]
                }
            }
        }
    ]
```

```
}
```

Any user signed in with one of these policies attached can't use the CloudWatch console to view any canary information. They can view canary information only for the canaries authorized by the policy and only by using the DescribeCanaries API or the describe-canaries AWS CLI command.

Creating a canary

> **Important**
>
> Make sure that you use Synthetics canaries to monitor only endpoints and APIs where you have ownership or permissions. Depending on the canary frequency settings, these endpoints might experience increased traffic.

When you use the CloudWatch console to create a canary, you can use a blueprint provided by CloudWatch to create your canary or you can write your own script. For more information, see Using canary blueprints.

You can also create a canary using CloudFormation if you are using your own script for the canary. For more information, see AWS::Synthetics::Canary in the *AWS CloudFormation User Guide*.

If you are writing your own script, you can use several functions that CloudWatch Synthetics has built into a library. For more information, see Synthetics runtime versions.

> **Note**
>
> When you create a canary, one of the layers created is a Synthetics layer prepended with `Synthetics`. This layer is owned by the Synthetics service account and contains the runtime code.

To create a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Canary**.

4. Choose one of the following:

   - To base your canary on a blueprint script, choose **Use a blueprint**, and then choose the type of canary you want to create. For more information about what each type of blueprint does, see Using canary blueprints.
   - To upload your own Node.js script to create a custom canary, choose **Upload a script**.

     You can then drag your script into the **Script** area or choose **Browse files** to navigate to the script in your file system.
   - To import your script from an S3 bucket, choose **Import from S3**. Under **Source location**, enter the complete path to your canary or choose **Browse S3**.

     You must have `s3:GetObject` and `s3:GetObjectVersion` permissions for the S3 bucket that you use. The bucket must be in the same AWS Region where you are creating the canary.
5. Under **Name**, enter a name for your canary. The name is used on many pages, so we recommend that you give it a descriptive name that distinguishes it from other canaries.
6. Under **Application or endpoint URL**, enter the URL that you want the canary to test. This URL must include the protocol (such as https://).

   If you want the canary to test an endpoint on a VPC, you must also enter information about your VPC later in this procedure.
7. To run this canary from multiple Regions, under **Locations**, select the additional Regions where the canary should also run. The current Region is automatically selected as the primary location. Regions are grouped by geographic area (Americas, Asia Pacific, EMEA) for convenience. For more information, see Multilocation canaries.
8. If you are using your own script for the canary, under **Lambda handler**, enter the entry point where you want the canary to start. For information on Lambda handler format, see Synthetics runtime versions .
9. Under **Script editor**, **Runtime version**, select a synthetics runtime version to execute the canary. For information on synthetics runtime versions, see Synthetics runtime versions .

   Under **Browser configuration**, you can enable the browser to test the canary. You must select at least one browser.
10. If you are using environment variables in your script, choose **Environment variables** and then specify a value for each environment variable defined in your script. For more information, see Environment variables.

11. Under **Schedule**, choose whether to run this canary just once, run it continuously using a rate expression, or schedule it using a cron expression.

    - When you use the CloudWatch console to create a canary that runs continuously, you can choose a rate anywhere between once a minute and after an hour.

    - For more information about writing a cron expression for canary scheduling, see Scheduling canary runs using cron.

12. (Optional) To set a timeout value for the canary, choose **Additional configuration** and then specify the timeout value. Make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

13. Under **Data retention**, specify how long to retain information about both failed and successful canary runs. The range is 1-455 days.

    This setting affects the range of information returned by GetCanaryRuns operations, as well as the range of information displayed in the Synthetics console.

    It does not affect the data stored in your Amazon S3 buckets, or logs or metrics that are published by the canary.

    Regardless of the canary's data retention period, the range of information displayed in console has certain limits. In the Synthetics console home view, the relative and absolute time range are limited to seven days. In the Synthetics console view for a specific canary, the relative time range is limited to seven days and the absolute time range is limited to 30 days.

14. Under **Data Storage**, select the Amazon S3 bucket to use to store the data from the canary runs. The bucket name can't contain a period (.). If you leave this blank, a default Amazon S3 bucket is used or created.

15. (Optional) By default, canaries store their artifacts on Amazon S3, and the artifacts are encrypted at rest using an AWS-managed AWS KMS key. You can use a different encryption option by choosing **Additional configuration** in the **Data Storage** section. You can then choose the type of key to use for encryption. For more information, see Encrypting canary artifacts.

16. Under **Access permissions**, choose whether to create an IAM role to run the canary or use an existing one.

If you have CloudWatch Synthetics create the role, it automatically includes all the necessary permissions. If you want to create the role yourself, see Required roles and permissions for canaries for information about the necessary permissions.

If you use the CloudWatch console to create a role for a canary when you create the canary, you can't re-use the role for other canaries, because these roles are specific to just one canary. If you have manually created a role that works for multiple canaries, you can use that existing role.

To use an existing role, you must have the `iam:PassRole` permission to pass that role to Synthetics and Lambda. You must also have the `iam:GetRole` permission.

> **Note**
>
> If role manager is enabled in your account, CloudWatch attaches the role for you, and the role options described here (for example the **Create new role** button) are replaced by a **Customize** option. To use a different role, choose **Customize**. For more information about IAM role creation, see IAM role creation in the *IAM User Guide*.

17. (Optional) Under **Alarms**, choose whether you want default CloudWatch alarms to be created for this canary. If you choose to create alarms, they are created with the following name convention:`Synthetics-Alarm-`*`canaryName`* `-`*`index`*

    `index` is a number representing each different alarm created for this canary. The first alarm has an index of 1, the second alarm has an index of 2, and so on.

18. (Optional) To have this canary test an endpoint that is on a VPC, choose **VPC settings**, and then do the following:

    a. Select the VPC that hosts the endpoint.

    b. Select one or more subnets on your VPC. You must select a private subnet because a Lambda instance can't be configured to run in a public subnet when an IP address can't be assigned to the Lambda instance during execution. For more information, see Configuring a Lambda Function to Access Resources in a VPC.

    c. Select one or more security groups on your VPC.

    d. To allow outbound IPv6 traffic for this canary, select **Allow IPv6 Traffic for dual-stack subnets**. This enables the canary to monitor IPv6-only and dual stack enabled endpoints over IPv6.

You can monitor endpoints external to your VPC by giving the canary internet access and configuring the VPC subnets appropriately. For more information, see Running a canary on a VPC.

e. If you selected replica Regions under **Locations**, you can optionally configure VPC settings for each replica Region. Under **Location VPC settings**, select a replica Region and choose a VPC, subnets, and security groups for that Region. VPC settings are not inherited from the primary canary.

If the endpoint is on a VPC, you must enable your canary to send information to CloudWatch and Amazon S3. For more information, see Running a canary on a VPC.

19. (Optional) Under **Tags**, add one or more key-value pairs as tags for this canary. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see Tagging your Amazon CloudWatch resources.

    If you want the tags that you apply to the canary to also be applied to the Lambda function that the canary uses, choose **Lambda function** under **Tag Replication**. If you choose this option, CloudWatch Synthetics will keep the tags on the canary and the Lambda function synchronized:

    - Synthetics will apply the same tags that you specify here to both your canary and your Lambda function.
    - If you later update the canary's tags and keep this option selected, Synthetics modifies the tags on your Lambda function to remain in sync with the canary.

    If you selected replica Regions under **Locations**, tags from the primary Region are not replicated to replica Regions. To add tags to a replica canary, navigate to the replica Region and add tags there.

20. (Optional) Under **Active tracing**, choose whether to enable active X-Ray tracing for this canary. Active tracing is only available for Puppeteer and Java runtimes. For more information, see Canaries and X-Ray tracing.

Resources that are created for canaries

When you create a canary, the following resources are created:

- An IAM role with the name `CloudWatchSyntheticsRole-`*`canary-name`* `-`*`uuid`* (if you use CloudWatch console to create the canary and specify for a new role to be created for the canary)
- An IAM policy with the name `CloudWatchSyntheticsPolicy-` *`canary-name`*`-`*`uuid`*.
- An S3 bucket with the name `cw-syn-results-`*`accountID`* `-`*`region`*.
- Alarms with the name `Synthetics-Alarm-`*`MyCanaryName`*, if you want alarms to be created for the canary.
- Lambda functions and layers, if you use a blueprint to create the canary. These resources have the prefix `cwsyn-`*`MyCanaryName`*.
- CloudWatch Logs log groups with the name `/aws/lambda/cwsyn-`*`MyCanaryName`* `-`*`randomId`*.

Using canary blueprints

This section provides details about each of the canary blueprints and the tasks each blueprint is best suited for. Blueprints are provided for the following canary types:

Topics

- Heartbeat monitoring
- API canary
- Broken link checker
- Visual monitoring blueprint
- Canary recorder
- GUI workflow builder
- Multi checks blueprint
- Creating multi checks blueprint canary

When you use a blueprint to create a canary, as you fill out the fields in the CloudWatch console, the **Script editor** area of the page displays the canary you are creating as a Node.js script. You can also edit your canary in this area to customize it further.

Heartbeat monitoring

Heartbeat scripts load the specified URL and store a screenshot of the page and an HTTP archive file (HAR file). They also store logs of accessed URLs.

You can use the HAR files to view detailed performance data about the web pages. You can analyze the list of web requests and catch performance issues such as time to load for an item.

If your canary uses the `syn-nodejs-puppeteer-3.1` or later runtime version, you can use the heartbeat monitoring blueprint to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

API canary

API canaries can test the basic Read and Write functions of a REST API. REST stands for *representational state transfer* and is a set of rules that developers follow when creating an API. One of these rules states that a link to a specific URL should return a piece of data.

Canaries can work with any APIs and test all types of functionality. Each canary can make multiple API calls.

In canaries that use runtime version `syn-nodejs-2.2` or later, the API canary blueprint supports multi-step canaries that monitor your APIs as HTTP steps. You can test multiple APIs in a single canary. Each step is a separate request that can access a different URL, use different headers, and use different rules for whether headers and response bodies are captured. By not capturing headers and response body, you can prevent sensitive data from being recorded.

Each request in an API canary consists of the following information:

- The *endpoint*, which is the URL that you request.
- The *method*, which is the type of request that is sent to the server. REST APIs support GET (read), POST (write), PUT (update), PATCH (update), and DELETE (delete) operations.
- The *headers*, which provide information to both the client and the server. They are used for authentication and providing information about the body content. For a list of valid headers, see HTTP Headers.
- The *data* (or *body*), which contains information to be sent to the server. This is used only for POST, PUT, PATCH, or DELETE requests.

> **Note**
>
> API canary blueprints are not supported by Playwright runtimes.

The API canary blueprint supports GET and POST methods. When you use this blueprint, you must specify headers. For example, you can specify **`Authorization`** as a **Key** and specify the necessary authorization data as the **Value** for that key.

If you are testing a POST request, you also specify the content to post in the **Data** field.

Integration with API Gateway

The API blueprint is integrated with Amazon API Gateway. This enables you to select an API Gateway API and stage from the same AWS account and Region as the canary, or to upload a Swagger template from API Gateway for cross-account and cross-Region API monitoring. You can then choose the rest of the details in the console to create the canary, instead of entering them from scratch. For more information about API Gateway, see What is Amazon API Gateway?

Using a private API

You can create a canary that uses a private API in Amazon API Gateway. For more information, see Creating a private API in Amazon API Gateway?

Broken link checker

The broken link checker collects all the links inside the URL that you are testing by using `document.getElementsByTagName('a')`. It tests only up to the number of links that you specify, and the URL itself is counted as the first link. For example, if you want to check all the links on a page that contains five links, you must specify for the canary to follow six links.

Broken link checker canaries created using the `syn-nodejs-2.0-beta` runtime or later support the following additional features:

- Provides a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- When viewing canary results, you can filter to see only the broken links and then fix the link based on the reason for failure.
- This version captures annotated source page screenshots for each link and highlights the anchor where the link was found. Hidden components are not annotated.
- You can configure this version to capture screenshots of both source and destination pages, just source pages, or just destination pages.
- This version fixes an issue in the earlier version where the canary script stops after the first broken link even when more links are scraped from the first page.

> **Note**
>
> Broken link checker blueprints are not supported by Playwright runtimes.

To update an existing canary using `syn-1.0` to use the new runtime, you must delete and re-create the canary. Updating an existing canary to the new runtime does not make these features available.

A broken link checker canary detects the following types of link errors:

- 404 Page Not Found
- Invalid Host Name
- Bad URL. For example, the URL is missing a bracket, has extra slashes, or uses the wrong protocol.
- Invalid HTTP response code.
- The host server returns empty responses with no content and no response code.
- The HTTP requests constantly time out during the canary's run.
- The host consistently drops connections because it is misconfigured or is too busy.

Visual monitoring blueprint

The visual monitoring blueprint includes code to compare screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium, or using Playwright runtimes.

The visual monitoring blueprint includes the following line of code in the default blueprint canary script, which enables visual monitoring.

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see Edit or delete a canary.

You can also change the canary run that is used as the baseline by using the `nextrun` or `lastrun` parameters or specifing a canary run ID in the UpdateCanary API.

When you use the visual monitoring blueprint, you enter the URL where you want the screenshot to be taken, and specify a difference threshold as a percentage. After the baseline run, future runs of the canary that detect a visual difference greater than that threshold trigger a canary failure. After the baseline run, you can also edit the canary to "draw" boundaries on the baseline screenshot that you want to ignore during the visual monitoring.

The visual monitoring feature is powered by the the ImageMagick open source software toolkit. For more information, see ImageMagick .

Canary recorder

With the canary recorder blueprint, you can use the CloudWatch Synthetics Recorder to record your click and type actions on a website and automatically generate a Node.js script that can be used to create a canary that follows the same steps. The CloudWatch Synthetics Recorder is a Google Chrome extension provided by Amazon. The canary recorder is not supported for canaries that use the Playwright runtime.

Credits: The CloudWatch Synthetics Recorder is based on the Headless recorder .

For more information, see Using the CloudWatch Synthetics Recorder for Google Chrome.

GUI workflow builder

The GUI Workflow Builder blueprint verifies that actions can be taken on your webpage. For example, if you have a webpage with a login form, the canary can populate the user and password fields and submit the form to verify that the webpage is working correctly.

When you use a blueprint to create this type of canary, you specify the actions that you want the canary to take on the webpage. The actions that you can use are the following:

- **Click**— Selects the element that you specify and simulates a user clicking or choosing the element.

  To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Verify selector**— Verifies that the specified element exists on the webpage. This test is useful for verifying that a previous action has causes the correct elements to populate the page.

  To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[class=]`.

- **Verify text**— Verifies that the specified string is contained within the target element. This test is useful for verifying that a previous action has caused the correct text to be displayed.

  To specify the element in a Node.js script, use a format such as `div[@id=]//h1` because this action uses the `waitForXPath` function in Puppeteer.

  To specify the element in a Python script, use xpath format such as `//*[@id=]` or //*[@class=] because this action uses the `implicitly_wait` function in Selenium.

- **Input text**— Writes the specified text in the target element.

  To specify the element to verify in a Node.js script, use `[id=]` or `a[class=]`.

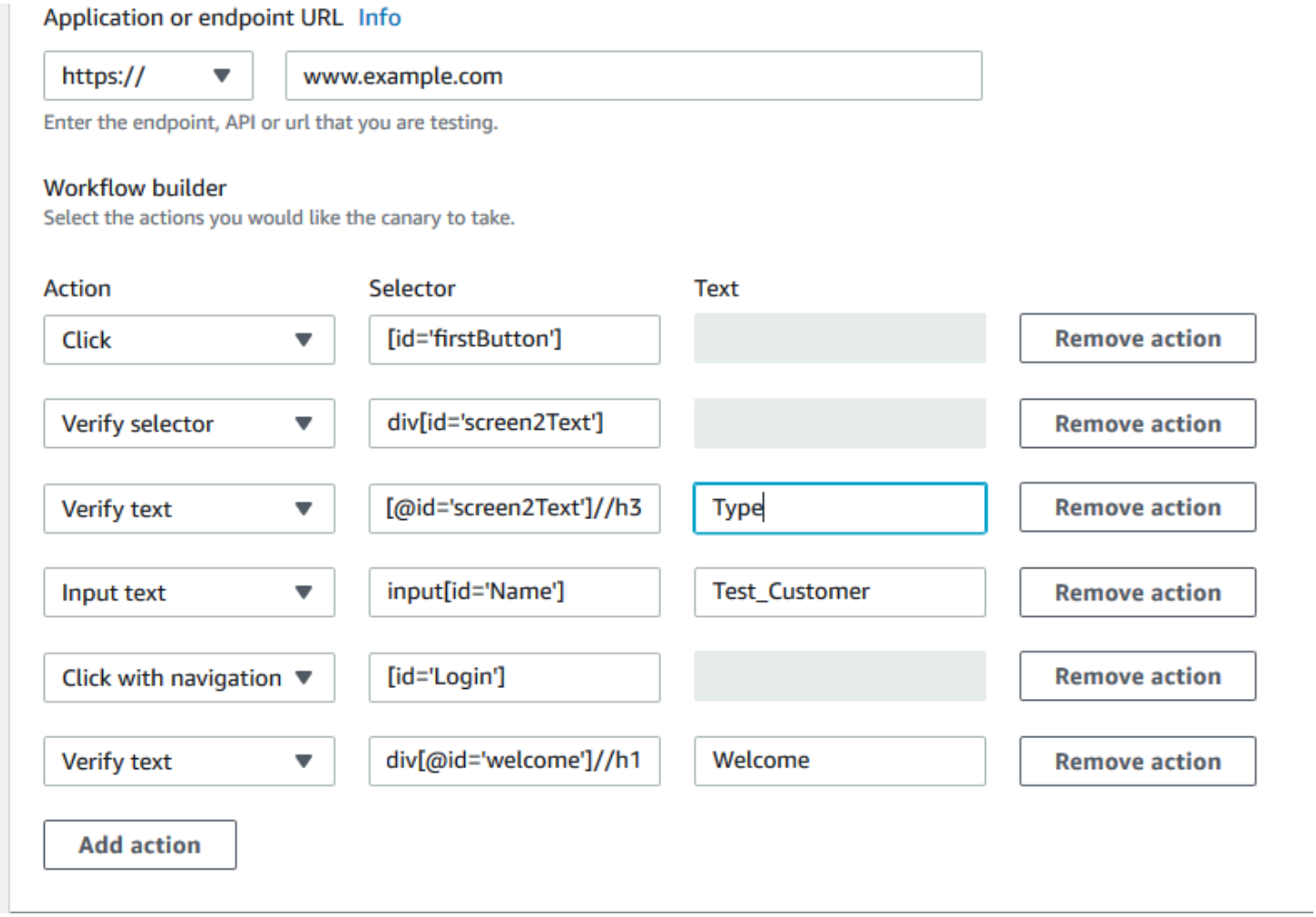
  To specify the element to verify in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

- **Click with navigation**— Waits for the whole page to load after choosing the specified element. This is most useful when you need to reload the page.

  To specify the element in a Node.js script, use `[id=]` or `a[class=]`.

  To specify the element in a Python script, use `xpath //*[@id=]` or `//*[@class=]`.

For example, the following blueprint uses Node.js. It clicks the **firstButton** on the specified URL, verifies that the expected selector with the expected text appears, inputs the name `Test_Customer` into the **Name** field, clicks the **Login** button, and then verifies that the login is successful by checking for the **Welcome** text on the next page.



GUI workflow canaries that use the following runtimes also provide a summary of the steps executed for each canary run. You can use the screenshots and error message associated with each step to find the root cause of failure.

- `syn-nodejs-2.0` or later
- `syn-python-selenium-1.0` or later

Multi checks blueprint

The Multi check blueprint simplifies canary creation. It reduces costs by using a simple JSON configuration that provides out-of-the-box functionality for performing HTTP, DNS, SSL, and TCP checks. You can configure up to 10 checks. Configure each check as a numerical step that runs sequentially, allowing for clear understanding of the canary flow.

Multi checks blueprints support:

- Basic HTTP requests, TCP requests, validate DNS records and monitor SSL certificates
- HTTP authentication methods such as Basic, API Key, OAuth, and Sigv4 integrated with Secrets Manager
- Assertions for each check

For more information, see Creating a canary.

Creating multi checks blueprint canary

Amazon CloudWatch Synthetics multi checks blueprint helps you create a Synthetics canary by providing a simple JSON configuration. You can save costs by bundling up to 10 different types of HTTP/DNS/SSL/TCP checks in a step-based sequential manner. Each check includes assertions that provide basic verification against a check result.

Multi checks canaries are designed for simple use cases that require only basic checks without a headless browser. For more complex use cases, review the other canary types that Amazon CloudWatch Synthetics provides.

Topics

- Prerequisites
- Limitations
- Packaging structure, JSON schema, and configuration settings
- Creating a multi check canary in AWS Management Console
- Creating a multi check canary using AWS Synthetics APIs
- Creating a multi check canary in CloudFormation
- Authentication configuration
- Troubleshooting

Prerequisites

- Must be using syn-nodejs-3.0+ in order to create a multi check canary
- When using Authentication and Secrets Manager configuration, you must ensure the canary ExecutionRoleArn allows for the permissions to access these secrets
- When using Authentication for Sigv4, you must ensure the canary ExecutionRoleArn allows for the permissions to access the related role

Limitations

- HTTP Response sizes cannot be greater than 1 MB
- Maximum of 10 defined variables.
- When using the JSON RFC, the Checks JSON may have duplicate fields provided however only the last sequential field will be used
- In the AWS Management Console, a multi check canary will default to showing multi check step metrics to readily identify the availability of each check. When checks are removed, this graph may still show the checks in the availability graph until the metric stops being active for at least 3 hours

Packaging structure, JSON schema, and configuration settings

The JSON Checks configuration that will be used for the canary must be named `blueprint-config.json`. The configuration must follow the schema and follow the instructions under Writing a JSON configuration for Node.js multi Checks blueprint.

Compress the `blueprint-config.json` into a ZIP file and provide it in one of the following creation workflows. When there is a `synthetics.json` configuration, then it is also compressed in the same ZIP file. The following is a zip file example called `multi-checks.zip`.

```
multi-checks.zip
### blueprint-config.json
### synthetics.json
```

Creating a multi check canary in AWS Management Console

1. Open the Amazon CloudWatch synthetics console.
2. Choose **Create Canary**.
3. Under **Use a blueprint**, choose **multi checks** .

   Under **Configure Checks**, you will see two tabs, **Checks** and **Canary configuration**.
4. Select the runtime version **syn-nodejs-3.0** or later.
5. Follow the procedure under Writing a JSON configuration for Node.js multi Checks blueprint to describe the check you want to perform. Alternatively, the console provides you a default JSON configuration that you can build on.

6. Choose **Create Canary**.

Creating a multi check canary using AWS Synthetics APIs

Use the `CreateCanary` API and within the `Code` parameter, provide the field/value `BlueprintTypes="multi-checks"` instead of `Handler`. When both `BlueprintTypes` and `Handler` are specified, a `ValidationException` is displayed. The runtime version provided must be `syn-nodejs-3.0` or later.

```
aws synthetics create-canary \
    --name my-multi-check-canary \
    --code ZipFile="ZIP_BLOB",BlueprintTypes="multi-checks" \
    --runtime-version syn-nodejs-3.0 \
    ...

// Or if you wanted to use S3 to provide your code.

aws synthetics create-canary \
    --name my-multi-check-canary \
    --code S3Bucket="my-code-bucket",S3Key="my-zip-code-key",BlueprintTypes="multi-
checks" \
    ...
```

Creating a multi check canary in CloudFormation

In your CloudFormation template for a multi check canary, within the `Code` parameter, provide the field/value `BlueprintTypes="multi-checks"` instead of `Handler`. When both `BlueprintTypes` and `Handler` are specified, a `ValidationException` is displayed. The runtime version provided must be `syn-nodejs-3.0 or later`.

An example template:

```
SyntheticsCanary:
    Type: 'AWS::Synthetics::Canary'
    Properties:
      Name: MyCanary
      RuntimeVersion: syn-nodejs-3.0
      Schedule: {Expression: 'rate(5 minutes)', DurationInSeconds: 3600}
```

```
      ...
      Code:
        S3Bucket: "my-code-bucket"
        S3Key: "my-zip-code-key"
        BlueprintTypes: ["multi-checks"]
      ...
```

Authentication configuration

When your canary makes HTTP requests to an authenticated endpoint, you can configure the steps of your blueprint canary to use one of four authentication types: Basic, API Key, OAuth Client Credentials, and SigV4. Rather than setting up request headers yourself, you can specify an authentication type in your blueprint definition. Synthetics follows the specified authentication type to populate the components of your HTTP request with the authentication information provided.

You specify an authentication type in your blueprint step with the Authentication section. You specify the authentication scheme you want to use, the properties required for your chosen authentication scheme, and Synthetics uses the information provided to construct an authentication header for your HTTP request.

Since storing secrets (such as passwords or API keys) in plain text is a security concern, Synthetics supports integration with AWS Secrets Manager. When you want to authenticate an HTTP request in a Synthetics blueprint canary, you can refer to the secret storing your authentication information and Synthetics handles retrieving the secret and caching it in your canary. This approach provides secrets to Synthetics while keeping your secrets securely stored, without specifying them in plain text in your blueprint configuration.

For more information about AWS Secrets Manager, see What is AWS Secrets Manager?

Basic authentication

Synthetics implements the Basic HTTP authentication scheme defined in RFC 7617. The process works as follows:

- A username and password pair is provided from the blueprint configuration.
- The user-pass is created by concatenating the username, a single colon (":") character, and the password.
- The user-pass is UTF-8 encoded, then converted into a base64-encoded string.

- This base64-encoded user-pass is provided in the "Authorization" header with the following format: Authorization: Basic {base64-encoded-user-pass}

For example, if the user agent wants to send the user-id "Aladdin" and password "open sesame", it uses the following header field: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Example configuration:

```
"Authentication": {
    "type": "BASIC",
    "username": MY_USERNAME, // Required
    "password": MY_PASSWORD // Required
}
```

API key authentication

You can provide an API key for authenticating your HTTP requests. When you use API key authentication, your provided API key is put into the "X-API-Key" HTTP header. If you have a custom resource that looks for API key headers in a header besides this one, you can optionally specify a different header name to have Synthetics put the API key into.

Example configuration:

```
"Authentication": {
    "type": "API_KEY",
    "apiKey": S0A1M2P3L4E5, // Required
    "header": X-Specific-Header // Optional, defaults to "X-API-Key"
}
```

SigV4 authentication

AWS SigV4 (Signature Version 4) is the AWS signing protocol for adding authentication information to AWS API requests. To make a SigV4-authenticated request, you need to specify the region and service you are making requests to, as well as an ARN (AWS Resource Name) identifying an IAM role you want the canary to assume when making this SigV4 request. Synthetics assumes the IAM role provided in the roleArn, and uses it to authenticate your AWS API request.

Example configuration:

```
"Authentication": {
    "type": "SIGV4",
    "region": us-west-2, // Required
    "service": s3, // Required
    "roleArn": arn:AWS:iam:12345678912:role/SampleRole // Required
}
```

SigV4 considerations

For Synthetics to assume the role that you provided in the SigV4 authentication section, the trust policy attached to that role must be configured to allow the canary to assume the provided roleArn. The AWS principal you need to trust is the role that your canary assumed through AWS STS. It takes the format `aws:sts::{account_running_the_canary}:assumed-role/<canary_name>/<assumed_role_name>` arn:.

For example, if you have a canary running in account 0123456789012, named test-canary, and the role it assumed was named canary-assume-role, then the trust policy needs to include this statement for the canary to assume the roleArn for SigV4 authentication correctly:

```
{
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:AWS:sts::123456789012:assumed-role/test-canary/"
    },
    "Action": "sts:AssumeRole"
}
```

OAuth client credentials

Synthetics implements the OAuth Client Credentials grant type as defined in RFC 6479 Section 4.4. If you want to make an HTTP request to an endpoint authenticated with a Bearer Token issued by an OAuth token endpoint, Synthetics can request and manage a bearer token on your behalf. When you use the OAuth scheme, Synthetics performs the following steps:

- Uses the Basic authentication scheme with the clientId and clientSecret to authenticate a request to the tokenUrl, the endpoint that issues bearer tokens
- If you provide the optional scope, audience, and resource parameters, they are included in the token request
- Uses the access token returned by the tokenUrl to authenticate your HTTP request
- Securely stores the refresh token returned from the tokenUrl for future token requests

Example configuration:

```
"Authentication": {
    "type": "OAUTH_CLIENT_CREDENTIALS",
    "tokenUrl": ..., // Required
    "clientId": ..., // Required
    "clientSecret": ..., // Required
    "scope": ..., // Optional
    "audience": ..., // Optional
    "resource": ..., // Optional
}
```

OAuth considerations

Synthetics refreshes OAuth tokens when a 401 or 407 response is returned.

AWS Secrets Manager integration

To avoid storing secret values (such as passwords or API keys) in plain text, Synthetics provides an integration with AWS Secrets Manager. You can reference an entire secret value in your blueprint configuration with the format `${AWS_SECRET:<secret_name>}`, or to reference a particular key `${AWS_SECRET:<secret_name>:<secret_key>}`.

For example, if you have a secret named login/basic-auth-credentials, storing a username and password with the following JSON structure:

```
{
    "username": "Aladdin",
    "password": "open sesame"
}
```

You can reference the username and password in your blueprint configuration as follows, and Synthetics handles retrieving the secret value and using its keys to authenticate your request:

```
"Authentication": {
    "type": "BASIC",
    "username": ${AWS_SECRET:login/basic-auth-credentials:username},
    "password": ${AWS_SECRET:login/basic-auth-credentials:password}
}
```

To allow Synthetics to retrieve the specified secret, the role ARN assumed by the canary needs to have both `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` permissions. If the secret is encrypted using a customer-managed key instead of the AWS managed key AWS/secretsmanager, then you also need kms:Decrypt permissions for that key.

Example permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": "arn:AWS:secretsmanager:us-
east-1:123456789012:secret:secretName-AbCdEf"
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:AWS:kms:us-east-1:123456789012:key/key-id"
        }
    ]
}
```

Troubleshooting

Common troubleshooting failures

The underlying code for multi check blueprint is written in Typescript. See the canary troubleshooting page for common failures: Troubleshooting a failed canary.

JSON check configuration syntax errors

When there are any syntactic errors related to the canary's JSON check configuration, the AWS Management Console will provide you a failure reason when you attempt to create the canary. If you are creating a canary using an API or CloudFormation, you will see the failure when the canary is executed for the first time. It is recommended to use the safe canary updates workflow for multi check canary. For more information, see Performing safe canary updates .

Networking or timeout failures

For any intermittent or consistent failures related to timeouts, network connection failures (for example, ENOTFOUND, ECONNRESET) consider turning on `DEBUG` logs such that the following run will provide more additional details on why the Checks are failing. To do so, provide the Environment Variable CW_SYNTHETICS_LOG_LEVEL: "DEBUG".

If there are still failures that you are unable to debug, consider reaching out to AWS Support or checking if any of the other provided Canary types from CloudWatch Synthetics more closely matches your use-case.

Using the CloudWatch Synthetics Recorder for Google Chrome

Amazon provides a CloudWatch Synthetics Recorder to help you create canaries more easily. The recorder is a Google Chrome extension.

The recorder records your click and type actions on a website and automatically generates a Node.js script that can be used to create a canary that follows the same steps.

After you start recording, the CloudWatch Synthetics Recorder detects your actions in the browser and converts them to a script. You can pause and resume the recording as needed. When you stop recording, the recorder produces a Node.js script of your actions, which you can easily copy with the **Copy to Clipboard** button. You can then use this script to create a canary in CloudWatch Synthetics.

Credits: The CloudWatch Synthetics Recorder is based on the Headless recorder .

Installing the CloudWatch Synthetics Recorder extension for Google Chrome

To use the CloudWatch Synthetics Recorder, you can start creating a canary and choose the **Canary Recorder** blueprint. If you do this when you haven't already downloaded the recorder, the CloudWatch Synthetics console provides a link to download it.

Alternatively, you can follow these steps to download and install the recorder directly.

To install the CloudWatch Synthetics Recorder

1. Using Google Chrome, go to this website: https://chrome.google.com/webstore/detail/cloudwatch-synthetics-rec/bhdnlmmgiplmbcdmkkdfplenecpegfno
2. Choose **Add to Chrome**, then choose **Add extension** .

Using the CloudWatch Synthetics Recorder for Google Chrome

To use the CloudWatch Synthetics Recorder to help you create a canary, you can choose **Create canary** in the CloudWatch console, and then choose **Use a blueprint**, **Canary Recorder**. For more information, see Creating a canary.

Alternatively, you can use the recorder to record steps without immediately using them to create a canary.

To use the CloudWatch Synthetics Recorder to record your actions on a website

1. Navigate to the page that you want to monitor.
2. Choose the Chrome extensions icon, and then choose **CloudWatch Synthetics Recorder**.
3. Choose **Start Recording**.
4. Perform the steps that you want to record. To pause recording, choose **Pause**.
5. When you are finished recording the workflow, choose **Stop recording** .
6. Choose **Copy to clipboard** to copy the generated script to your clipboard. Or, if you want to start over, choose **New recording**.
7. To create a canary with the copied script, you can paste your copied script into the recorder blueprint inline editor, or save it to an Amazon S3 bucket and import it from there.
8. If you're not immediately creating a canary, you can save your recorded script to a file.

Known limitations of the CloudWatch Synthetics Recorder

The CloudWatch Synthetics Recorder for Google Chrome currently has the following limitations.

- HTML elements that don't have IDs will use CSS selectors. This can break canaries if the webpage structure changes later. We plan to provide some configuration options (such as using data-id) around this in a future version of the recorder.
- The recorder doesn't support actions such as double-click or copy/paste, and doesn't support key combinations such as CMD+0.
- To verify the presence of an element or text on the page, users must add assertions after the script is generated. The recorder doesn't support verifying an element without performing any action on that element. This is similar to the "Verify text" or "Verify element" options in the canary workflow builder. We plan to add some assertions support in a future version of the recorder.
- The recorder records all actions in the tab where the recording is initiated. It doesn't record pop-ups (for instance, to allow location tracking) or navigation to different pages from pop-ups.

Synthetics runtime versions

When you create or update a canary, you choose a Synthetics runtime version for the canary. A Synthetics runtime is a combination of the Synthetics code that calls your script handler, and the Lambda layers of bundled dependencies.

CloudWatch Synthetics currently supports runtimes that use Node.js, Python, or Java languages. The frameworks supported are Puppeteer, Playwright, and Selenium.

We recommend that you always use the most recent runtime version for your canaries, to be able to use the latest features and updates made to the Synthetics library.

Please note: whenever you run a canary to use the new version of the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically moved to the same version of NodeJS that the Synthetics runtime supports.

Topics

- Runtime versions using Java
- Runtime versions using Node.js and Playwright
- Runtime versions using Node.js and Puppeteer
- Runtime versions using Python and Selenium Webdriver

- Runtime versions using Node.js
- Runtime versions support policy
- Runtime versions update

Runtime versions using Java

The following section contains information about the CloudWatch Synthetics runtime versions for Java. This runtime does not have any browser or framework included.

The naming convention for these runtime versions is `syn-`*`language`*`-`*`majorversion`*`.`*`minorversion`*.

syn-java-1.0

Major dependencies:

- AWS Lambda runtime Java 21

Features

- *CloudWatch Logs integration* - You can query and filter for logs through the CloudWatch Synthetics console. Each log message contains unique `canaryRunId`, making it easy to search for logs for a particular canary run.
- *Metrics* - You can monitor canary run success percentage and duration through CloudWatch metrics. You can also configure alarms to alert you when canaries detect issues.
- *Canary artifacts* - Each canary run uploads a detailed report corresponding to the run and the steps in the run which can be accessed through Amazon S3.
- *Support for traces* - You can emit traces for all the requests made by the canary through X-Ray. Each canary run is associated with one trace Id.

Runtime versions using Node.js and Playwright

The following sections contain information about the CloudWatch Synthetics runtime versions for Node.js and Playwright. Playwright is an open-source automation library for browser testing. For more information about Playwright, see https://playwright.dev/

The naming convention for these runtime versions is `syn-`*`language`*`-`*`framework`*`-`*`majorversion`*`.`*`minorversion`*.

Important

Starting Synthetics `syn-nodejs-playwright-5.1` and later, Synthetics runtime uses the new namespace. Please migrate the canary script to use the new namespace. Legacy namespace will be deprecated in a future release.

- @amzn/synthetics-playwright → @aws/synthetics-playwright

syn-nodejs-playwright-9.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.63.0
- Playwright/test version 1.63.0
- Chromium version 153.0.8010.36
- Firefox version 155.0

Changes in syn-nodejs-playwright-9.0

- Upgrade `Chromium` to 153.0.8010.36 to address the following CVEs:
  - CVE-2026-85046
  - CVE-2026-87491
- Upgrade `axios` to 1.18.0 to address the following CVEs:
  - CVE-2026-67312
  - CVE-2026-67313
  - CVE-2026-67314
  - CVE-2026-67315
  - CVE-2026-67316
  - CVE-2026-67317
  - CVE-2026-67318
  - CVE-2026-67319
  - CVE-2026-67320

- CVE-2026-67321

For more information, see the following:

- Playwright release notes on the Playwright website
- Playwright API reference on the Playwright website

Previous runtime versions for Node.js and Playwright

The following earlier runtime versions for Node.js and Playwright are still supported.

syn-nodejs-playwright-8.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.61.1
- Playwright/test version 1.61.1
- Chromium version 150.0.7871.24
- Firefox version 151.0

Changes in syn-nodejs-playwright-8.0

- Upgrade `Chromium` to 150.0.7871.24 to address the following CVEs:
  - CVE-2026-11645
- Upgrade `ImageMagick` to 7.1.2-27 to address the following CVEs:
  - CVE-2026-28494
  - CVE-2026-28691
  - CVE-2026-28693
  - CVE-2026-30883
  - CVE-2026-30929
  - CVE-2026-30931
  - CVE-2026-32636
  - CVE-2026-33900

- CVE-2026-33901
- CVE-2026-33905
- CVE-2026-33908

syn-nodejs-playwright-7.1

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.59.1
- Playwright/test version 1.59.1
- Chromium version 147.0.7727.15
- Firefox version 148.0.2

Changes in syn-nodejs-playwright-7.1

- Fix bug where HTTP headers with array values were not being captured properly.
- Upgrade `ws` to 8.20.1 to address the following CVEs:
  - CVE-2026-45736
- Upgrade `fast-xml-parser` to 5.7.2 to address the following CVEs:
  - CVE-2026-41650

syn-nodejs-playwright-7.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.59.1
- Playwright/test version 1.59.1
- Chromium version 147.0.7727.15
- Firefox version 148.0.2

Changes in syn-nodejs-playwright-7.0

- Multilocation canaries – Run the same canary across multiple AWS Regions from a single point of management.
- Upgrade `ImageMagick` to 7.1.2-15 to address the following CVEs:
  - CVE-2023-34152
  - CVE-2025-53014
  - CVE-2025-53101
  - CVE-2025-57807
  - CVE-2026-22770
  - CVE-2026-23876
  - CVE-2026-25897
  - CVE-2026-25898
  - CVE-2026-25968
  - CVE-2026-25971
  - CVE-2026-25983
  - CVE-2026-25986
  - CVE-2026-25987
  - CVE-2026-26284

syn-nodejs-playwright-6.1

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.59.1
- Playwright/test version 1.59.1
- Chromium version 147.0.7727.15
- Firefox version 148.0.2

Changes in syn-nodejs-playwright-6.1

- Updated Playwright and browser versions.
- Upgrade `fast-xml-parser` to 5.5.7 to address the following CVEs:
  - CVE-2026-27942

  - CVE-2026-33036
- Upgrade `Chromium` to 147.0.7727.15 to address the following CVEs:
  - CVE-2026-3909
  - CVE-2026-3910
  - CVE-2026-5281

syn-nodejs-playwright-6.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.58.2
- Playwright/test version 1.58.2
- Chromium version 145.0.7632.77
- Firefox version 146.0.1

Changes in syn-nodejs-playwright-6.0

- Applied security patches and updated Playwright and browser versions.

syn-nodejs-playwright-5.1

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.57.0
- Playwright/test version 1.57.0
- Chromium version 143.0.7499.169
- Firefox version 142.0.1

Changes in syn-nodejs-playwright-5.1

- Synthetics runtime namespace migration.
- Type definition is available in npm Registry. Please make sure the type definition package version matches your canary's runtime version.

syn-nodejs-playwright-5.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.57.0
- Playwright/test version 1.57.0
- Chromium version 143.0.7499.4
- Firefox version 142.0.1

Changes in syn-nodejs-playwright-5.0

- Applied security patches and updated Playwright and browser versions.

syn-nodejs-playwright-4.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x
- Playwright version 1.55.0
- Playwright/test version 1.55.0
- Chromium version 140.0.7339.16
- Firefox version 141.0

Changes in syn-nodejs-playwright-4.0

- Applied security patches and updated Playwright and browser versions.

syn-nodejs-playwright-3.0

Major dependencies:

- AWS Lambda runtime Node.js 20.x
- Playwright version 1.53.0
- Playwright/test version 1.53.0
- Chromium version 138.0.7204.168

Changes in syn-nodejs-playwright-3.0

- Multi-browser support – You can now run your nodejs puppeteer canaries in either Firefox or Chrome
- Support for visual monitoring

syn-nodejs-playwright-2.0

Major dependencies:

- AWS Lambda runtime Node.js 20.x
- Playwright version 1.49.1
- Playwright/test version 1.49.1
- Chromium version 131.0.6778.264

Changes in syn-nodejs-playwright-2.0

- The mismatch between total duration and sum of timings for a given request in HAR file is fixed.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

syn-nodejs-playwright-1.0

Major dependencies:

- AWS Lambda runtime Node.js 20.x
- Playwright version 1.44.1
- Playwright/test version 1.44.1
- Chromium version 126.0.6478.126

Features:

- **PlayWright support** – You can write canary scripts by using the Playwright automation framework. You can bring your existing Playwright scripts to run as canaries, and enhance them with AWS monitoring capabilities.

- **CloudWatch Logs integration** – You can query and filter for logs through the CloudWatch Synthetics console. Each log message contains unique `canaryRunId`, making it easy to search for logs for a particular canary run.
- **Metrics and canary artifacts** – You can monitor canary run pass rate through CloudWatch metrics, and configure alarms to alert you when canaries detect issues.
- **Screenshots and steps association** – You can capture screenshots using native Playwright functionality to visualize the stages of a canary script on each run. Screenshots are automatically associated with canary steps, and are uploaded to Amazon S3 buckets.
- **Multiple tabs**– You can create canaries that open multiple browser tabs, and access screenshots from each tab. You can create multi-tab and multi-step user workflows in Synthetics.

Runtime versions using Node.js and Puppeteer

The first runtime version for Node.js and Puppeteer was named `syn-1.0`. Later runtime versions have the naming convention `syn-`*`language -majorversion.minorversion`*. Starting with `syn-nodejs-puppeteer-3.0`, the naming convention is `syn-`*`language-framework-majorversion .minorversion`*

An additional `-beta` suffix shows that the runtime version is currently in a beta preview release.

Runtime versions with the same major version number are backward compatible.

The Lambda code in a canary is configured to have a maximum memory of 1 GB. Each run of a canary times out after a configured timeout value. If no timeout value is specified for a canary, CloudWatch chooses a timeout value based on the canary's frequency. If you configure a timeout value, make it no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

> **Important**
>
> Starting Synthetics `syn-nodejs-puppeteer-13.1` and later, Synthetics runtime uses the new namespace. Please migrate the canary script to use new namespaces. Legacy namespaces will be deprecated in a future release.
>
> - Synthetics → @aws/synthetics-puppeteer
> - SyntheticsLink → @aws/synthetics-link
> - SyntheticsLogger → @aws/synthetics-logger

- SyntheticsLogHelper → @aws/synthetics-log-helper
- BrokenLinkCheckerReport → @aws/synthetics-broken-link-checker-report

Important

Synthetics runtime `syn-nodejs-puppeteer-11.0` and later versions support only the following step-level configuration overrides:

- `screenshotOnStepStart`
- `screenshotOnStepSuccess`
- `screenshotOnStepFailure`
- `stepSuccessMetric`
- `stepDurationMetric`
- `continueOnStepFailure/continueOnHttpStepFailure`
- `stepsReport`

syn-nodejs-puppeteer-18.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 25.11.0
- Chromium version 153.0.8010.36
- Firefox version 155.0.1

Changes in syn-nodejs-puppeteer-18.0

- Upgrade `Chromium` to 153.0.8010.36 to address the following CVEs:
  - CVE-2026-85046
  - CVE-2026-87491
- Upgrade `axios` to 1.18.0 to address the following CVEs:
  - CVE-2026-67312

- CVE-2026-67313
- CVE-2026-67314
- CVE-2026-67315
- CVE-2026-67316
- CVE-2026-67317
- CVE-2026-67318
- CVE-2026-67319
- CVE-2026-67320
- CVE-2026-67321

For more information, see the following:

- Puppeteer changelog on the Puppeteer website
- Puppeteer API reference on the Puppeteer website

Previous runtime versions for Node.js and Puppeteer

The following earlier runtime versions for Node.js and Puppeteer are still supported.

syn-nodejs-puppeteer-17.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 25.2.1
- Chromium version 150.0.7871.24
- Firefox version 152.0.2

Changes in syn-nodejs-puppeteer-17.0

- Upgrade `Chromium` to 150.0.7871.24 to address the following CVEs:
  - CVE-2026-11645
- Upgrade `ws` to 8.21.0 to address the following CVEs:
  - CVE-2026-48779

- Upgrade `ImageMagick` to 7.1.2-27 to address the following CVEs:
  - CVE-2026-28494
  - CVE-2026-28691
  - CVE-2026-28693
  - CVE-2026-30883
  - CVE-2026-30929
  - CVE-2026-30931
  - CVE-2026-32636
  - CVE-2026-33900
  - CVE-2026-33901
  - CVE-2026-33905
  - CVE-2026-33908

syn-nodejs-puppeteer-16.1

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.42.0
- Chromium version 147.0.7727.57
- Firefox version 147.0.4

Changes in syn-nodejs-puppeteer-16.1

- Fix bug where HTTP headers with array values were not being captured properly.
- Upgrade `ip-address` to 10.1.1 to address the following CVEs:
  - CVE-2026-42338
- Upgrade `ws` to 8.20.1 to address the following CVEs:
  - CVE-2026-45736

syn-nodejs-puppeteer-16.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.42.0
- Chromium version 147.0.7727.57
- Firefox version 147.0.4

Changes in syn-nodejs-puppeteer-16.0

- Multilocation canaries – Run the same canary across multiple AWS Regions from a single point of management.
- Upgrade `ImageMagick` to 7.1.2-15 to address the following CVEs:
  - CVE-2023-34152
  - CVE-2025-53014
  - CVE-2025-53101
  - CVE-2025-57807
  - CVE-2026-22770
  - CVE-2026-23876
  - CVE-2026-25897
  - CVE-2026-25898
  - CVE-2026-25968
  - CVE-2026-25971
  - CVE-2026-25983
  - CVE-2026-25986
  - CVE-2026-25987
  - CVE-2026-26284

syn-nodejs-puppeteer-15.1

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.42.0
- Chromium version 147.0.7727.57
- Firefox version 147.0.4

Changes in syn-nodejs-puppeteer-15.1

- Updated Puppeteer and browser versions.
- Upgrade `basic-ftp` to 5.3.1 to address the following CVEs:
  - CVE-2026-39983
  - CVE-2026-41324
  - GHSA-6v7q-wjvx-w8wg
  - GHSA-rpmf-866q-6p89
- Upgrade `Chromium` to 147.0.7727.57 to address the following CVEs:
  - CVE-2026-3909
  - CVE-2026-3910
  - CVE-2026-5281

syn-nodejs-puppeteer-15.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.37.5
- Chromium version 145.0.7632.77
- Firefox version 147.0.4

Changes in syn-nodejs-puppeteer-15.0

- Applied security patches and updated Puppeteer and browser versions.
- Fixed bug where continueOnHttpStepFailure was not being honored, causing canary runs to be incorrectly marked as successful despite HTTP step failures occurring.

syn-nodejs-puppeteer-14.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.34.0

- Chromium version 143.0.7499.169
- Firefox version 146.x

Changes in syn-nodejs-puppeteer-14.0

- Applied security patches and updated Puppeteer and browser versions.

syn-nodejs-puppeteer-13.1

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.25.0
- Chromium version 142.0.7444.175
- Firefox version 145.x

Changes in syn-nodejs-puppeteer-13.1

- Synthetics runtime namespace migration.
- Type definitions are available in npm Registry. Please make sure the type definition package version matches your canary's runtime version.
  - @aws/synthetics-puppeteer
  - @aws/synthetics-link
  - @aws/synthetics-broken-link-checker-report
  - @aws/synthetics-log-helper
  - @aws/synthetics-logger

syn-nodejs-puppeteer-13.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.25.0
- Chromium version 142.0.7444.175
- Firefox version 145.x

Changes in syn-nodejs-puppeteer-13.0

- Applied security patches and updated Puppeteer and browser versions.
- Bug fix – Fixed intermittent runtime extension crash issue caused by concurrent map access

syn-nodejs-puppeteer-12.0

Major dependencies:

- Lambda runtime Node.js 22.x
- Puppeteer-core version 24.22.1
- Chromium version 140.0.7339.185
- Firefox version 143.0.1

Changes in syn-nodejs-puppeteer-12.0

- Applied security patches and updated Puppeteer and browser versions.
- Bug fix for Restricted header redaction – Fixed an issue where in some situations restricted headers were not being redacted in executeHttpStep(). Behavior is now consistent with Puppeteer 10.0.
- Bug fix for includeResponseBody configuration – Fixed an issue where HAR file generation can misapply the includeResponseBody configuration setting in certain situations. HAR now ensures response bodies are excluded when setting is configured.
- Request capture lifecycle fixed – Fixed an issue where in some situations the HTTP request capturer may cause continuous aggregation of requests. Recording now terminates correctly after each step execution.

syn-nodejs-puppeteer-11.0

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 24.15.0
- Chromium version 138.0.7204.168

Changes in syn-nodejs-puppeteer-11.0

- Multi-browser support – You can now run Node.js Puppeteer canaries in either Firefox or Chrome
- Simplified packaging – Package scripts directly under root without using the Node.js/node_modules directory structure
- Screenshot integration – Capture screenshots using native Puppeteer functions to visualize canary script stages. Synthetics automatically associates screenshots with canary steps and uploads them to Amazon S3
- Enhanced log querying – Query and filter logs through the CloudWatch Insights console. Each log message includes a unique `canaryRunId` for easier searching
- Configuration file support – Define and update Synthetics settings using a synthetics.json file. This separation of configuration from script logic improves maintenance and reusability
- Multiple tabs support – Create canaries that open multiple browser tabs and access screenshots from each tab. Build multi-tab and multi-step user workflows in Synthetics
- Security fixes
- Visual monitoring bug fixes
- Added support for structured JSON logging with configurable log levels – Logs are now emitted in JSON format to enable easier parsing and querying in CloudWatch. Log level is configurable (for example, DEBUG, INFO, TRACE) through environment variables allowing users to control verbosity based on their needs
- Support for ES syntax

syn-nodejs-puppeteer-10.0

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 24.2.0
- Chromium version 131.0.6778.264

Changes in syn-nodejs-puppeteer-10.0

- The bug related to closing the browser that took excessively long is fixed.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

syn-nodejs-puppeteer-9.1

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.12.1
- Chromium version 126.0.6478.126

Changes in syn-nodejs-puppeteer-9.1 – Bug fixes related to date ranges and pending requests in HAR files are fixed.

syn-nodejs-puppeteer-9.0

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.12.1
- Chromium version 126.0.6478.126

Changes in syn-nodejs-puppeteer-9.0 – The bug fix to enable visual monitoring capabilities is fixed.

syn-nodejs-puppeteer-8.0

> **Warning**
>
> Because of a bug, the `syn-nodejs-puppeteer-8.0` runtime doesn't support visual monitoring in canaries. Upgrade to syn-nodejs-puppeteer-9.0 for the bug fix for visual monitoring.

> **Important**
>
> Lambda Node.js 18 and later runtimes use AWS SDK for JavaScript V3. If you need to migrate a canary from an earlier runtime, follow the aws-sdk-js-v3 Migration Workshop on GitHub. For more information about AWS SDK for JavaScript version 3, see this blog post.

Major dependencies:

- Lambda runtime Node.js 20.x
- Puppeteer-core version 22.10.0
- Chromium version 125.0.6422.112

Updates in syn-nodejs-puppeteer-8.0:

- **Support for two-factor authentication**
- **Bug fixes** related to some service clients losing data in Node.js SDK V3 responses is fixed.

Deprecated runtime versions for Node.js and Puppeteer

The following runtimes for Node.js and Puppeteer have been deprecated. For information about runtime deprecation dates, see CloudWatch Synthetics runtime deprecation dates.

syn-nodejs-puppeteer-7.0

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 21.9.0
- Chromium version 121.0.6167.139

Code size:

The size of code and dependencies that you can package into this runtime is 80 MB.

Updates in syn-nodejs-puppeteer-7.0:

- **Updated versions of the bundled libraries in Puppeteer and Chromium**— The Puppeteer and Chromium dependencies are updated to new versions.

> **Important**
>
> Moving from Puppeteer 19.7.0 to Puppeteer 21.9.0 introduces breaking changes regarding testing and filters. For more information, see the **BREAKING CHANGES** sections in puppeteer: v20.0.0 and puppeteer-core: v21.0.0.

Recommended upgrade to AWS SDK v3

The Lambda nodejs18.x runtime doesn't support AWS SDK v2. You strongly recommend that you migrate to AWS SDK v3.

syn-nodejs-puppeteer-6.2

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Changes in syn-nodejs-puppeteer-6.2:

- **Updated versions of the bundled libraries in Chromium**
- **Ephemeral storage monitoring**— This runtime adds ephemeral storage monitoring in customer accounts.
- **Bug fixes**

syn-nodejs-puppeteer-6.1

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-6.1:

- **Stability improvements**— Added auto-retry logic for handling intermittent Puppeteer launch errors.
- **Dependency upgrades**— Upgrades for some third-party dependency packages.
- **Canaries without Amazon S3 permissions**— Bug fixes, such that canaries that don't have any Amazon S3 permissions can still run. These canaries with no Amazon S3 permissions won't be able to upload screenshots or other artifacts to Amazon S3. For more information about permissions for canaries, see Required roles and permissions for canaries.

> **Important**
>
> IMPORTANT: The included AWS SDK for JavaScript v2 dependency will be removed and updated to use AWS SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue referencing and using the included AWS SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-6.0

Major dependencies:

- Lambda runtime Node.js 18.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-6.0:

- **Dependency upgrade**— The Node.js dependency is upgraded to 18.x.
- **Intercept mode support**— Puppeteer cooperative intercept mode support was added to the Synthetics canary runtime library.
- **Tracing behavior change**— Changed default tracing behavior to trace only fetch and xhr requests, and not trace resource requests. You can enable the tracing of resource requests by configuring the `traceResourceRequests` option.
- **Duration metric refined**— The `Duration` metric now excludes the operation time the canary uses to upload artifacts, take screenshots, and generate CloudWatch metrics. `Duration` metric values are reported to CloudWatch, and you can also see them in the Synthetics console.
- **Bug fix**— Clean up core dump generated when Chromium crashes during a canary run.

> **Important**
>
> IMPORTANT: The included AWS SDK for JavaScript v2 dependency will be removed and updated to use AWS SDK for JavaScript v3 in a future runtime release. When that happens, you can update your canary code references. Alternatively, you can continue

referencing and using the included AWS SDK for JavaScript v2 dependency by adding it as a dependency to your source code zip file.

syn-nodejs-puppeteer-5.2

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-5.2:

- **Updated versions of the bundled libraries in Chromium**
- **Bug fixes**

syn-nodejs-puppeteer-5.1

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Bug fixes in syn-nodejs-puppeteer-5.1:

- **Bug fix**— This runtime fixes a bug in `syn-nodejs-puppeteer-5.0` where the HAR files created by the canaries were missing request headers.

syn-nodejs-puppeteer-5.0

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 19.7.0
- Chromium version 111.0.5563.146

Updates in syn-nodejs-puppeteer-5.0:

- **Dependency upgrade**— The Puppeteer-core version is updated to 19.7.0. The Chromium version is upgraded to 111.0.5563.146.

> **Important**
>
> The new Puppeteer-core version is not completely backward-compatible with previous versions of Puppeteer. Some of the changes in this version can cause existing canaries that use deprecated Puppeteer functions to fail. For more information, see the breaking changes in the change logs for Puppeteer-core versions 19.7.0 through 6.0, in Puppeteer change logs.

syn-nodejs-puppeteer-4.0

Major dependencies:

- Lambda runtime Node.js 16.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-4.0:

- **Dependency upgrade**— The Node.js dependency is updated to 16.x.

syn-nodejs-puppeteer-3.9

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 14.x

- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.9:

- **Dependency upgrades**— Upgrades some third-party dependency packages.

syn-nodejs-puppeteer-3.8

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.8:

- **Profile cleanup**— Chromium profiles are now cleaned up after each canary run.

Bug fixes in syn-nodejs-puppeteer-3.8:

- **Bug fixes**— Previously, visual monitoring canaries would sometimes stop working properly after a run with no screenshots. This is now fixed.

syn-nodejs-puppeteer-3.7

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.7:

- **Logging enhancement**— The canary will upload logs to Amazon S3 even if it times out or crashes.
- **Lambda layer size reduced**— The size of the Lambda layer used for canaries is reduced by 34%.

Bug fixes in syn-nodejs-puppeteer-3.7:

- **Bug fixes**— Japanese, Simplified Chinese, and Traditional Chinese fonts will render properly.

syn-nodejs-puppeteer-3.6

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.6:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-nodejs-puppeteer-3.5

> **Important**
>
> This runtime version was deprecated on January 8, 2024. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 14.x
- Puppeteer-core version 5.5.0
- Chromium version 92.0.4512

Updates in syn-nodejs-puppeteer-3.5:

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-nodejs-puppeteer-3.4

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.4:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.

- **Expanded HAR file information**— You can now see bad, pending, and incomplete requests in the HAR files produced by canaries.

syn-nodejs-puppeteer-3.3

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.3:

- **More options for artifact encryption**— For canaries using this runtime or later, instead of using an AWS managed key to encrypt artifacts that the canary stores in Amazon S3, you can choose to use an AWS KMS customer managed key or an Amazon S3-managed key. For more information, see Encrypting canary artifacts.

syn-nodejs-puppeteer-3.2

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.2:

- **visual monitoring with screenshots**— Canaries using this runtime or later can compare a screenshot taken during a run with a baseline version of the same screenshot. If the screenshots are more different than a specified percentage threshold, the canary fails. For more information, see Visual monitoring or Visual monitoring blueprint.
- **New functions regarding sensitive data** You can prevent sensitive data from appearing in canary logs and reports. For more information, see SyntheticsLogHelper class.
- **Deprecated function** The `RequestResponseLogHelper` class is deprecated in favor of other new configuration options. For more information, see RequestResponseLogHelper class.

syn-nodejs-puppeteer-3.1

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.1:

- **Ability to configure CloudWatch metrics**— With this runtime, you can disable the metrics that you do not require. Otherwise, canaries publish various CloudWatch metrics for each canary run.
- **Screenshot linking**— You can link a screenshot to a canary step after the step has completed. To do this, you take the screenshot by using the **takeScreenshot** method, using the name of the

step that you want to associate the screenshot with. For example, you might want to perform a step, add a wait time, and then take the screenshot.

- **Heartbeat monitor blueprint can monitor multiple URLs**— You can use the heartbeat monitoring blueprint in the CloudWatch console to monitor multiple URLs and see the status, duration, associated screenshots, and failure reason for each URL in the step summary of the canary run report.

syn-nodejs-puppeteer-3.0

> **Important**
>
> This runtime version was deprecated on November 13, 2022. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 12.x
- Puppeteer-core version 5.5.0
- Chromium version 88.0.4298.0

Updates in syn-nodejs-puppeteer-3.0:

- **Upgraded dependencies**— This version uses Puppeteer version 5.5.0, Node.js 12.x, and Chromium 88.0.4298.0.
- **Cross-Region bucket access**— You can now specify an S3 bucket in another Region as the bucket where your canary stores its log files, screenshots, and HAR files.
- **New functions available**— This version adds library functions to retrieve the canary name and the Synthetics runtime version.

  For more information, see Synthetics class.

syn-nodejs-2.2

This section contains information about the `syn-nodejs-2.2` runtime version.

> **Important**
>
> This runtime version was deprecated on May 28, 2021. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Changes in syn-nodejs-2.2:

- **Monitor your canaries as HTTP steps**— You can now test multiple APIs in a single canary. Each API is tested as a separate HTTP step, and CloudWatch Synthetics monitors the status of each step using step metrics and the CloudWatch Synthetics step report. CloudWatch Synthetics creates `SuccessPercent` and `Duration` metrics for each HTTP step.

  This functionality is implemented by the **executeHttpStep(stepName, requestOptions, callback, stepConfig)** function. For more information, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).

  The API canary blueprint is updated to use this new feature.
- **HTTP request reporting**— You can now view detailed HTTP requests reports which capture details such as request/response headers, response body, status code, error and performance timings, TCP connection time, TLS handshake time, first byte time, and content transfer time. All HTTP requests which use the HTTP/HTTPS module under the hood are captured here. Headers and response body are not captured by default but can be enabled by setting configuration options.
- **Global and step-level configuration**— You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key/value pairs to enable or disable certain options.

  For more information, see SyntheticsConfiguration class.

- **Continue on step failure configuration**— You can choose to continue canary execution when a step fails. For the `executeHttpStep` function, this is turned on by default. You can set this option once at global level or set it differently per-step.

syn-nodejs-2.1

> **Important**
>
> This runtime version was deprecated on May 28, 2021. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Updates in syn-nodejs-2.1:

- **Configurable screenshot behavior**— Provides the ability to turn off the capturing of screenshots by UI canaries. In canaries that use previous versions of the runtimes, UI canaries always capture screenshots before and after each step. With `syn-nodejs-2.1`, this is configurable. Turning off screenshots can reduce your Amazon S3 storage costs, and can help you comply with HIPAA regulations. For more information, see SyntheticsConfiguration class.
- **Customize the Google Chrome launch parameters** You can now configure the arguments used when a canary launches a Google Chrome browser window. For more information, see launch(options).

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

syn-nodejs-2.0

> **Important**
>
> This runtime version was deprecated on May 28, 2021. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Updates in syn-nodejs-2.0:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Support for X-Ray active tracing.** When a canary has tracing enabled, X-Ray traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the X-Ray Trace Map, even when they don't send requests to other services or applications that have tracing enabled. For more information, see Canaries and X-Ray tracing.
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.
- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. With this runtime version, these metrics are reported only for UI canaries, and are not reported for API canaries. They are also reported for API canaries starting with runtime version `syn-nodejs-puppeteeer-2.2`.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.

- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

There can be a small increase in canary duration when using syn-nodejs-2.0 or later, compared to earlier versions of the canary runtimes.

syn-nodejs-2.0-beta

> **Important**
>
> This runtime version was deprecated on February 8, 2021. For more information, see Runtime versions support policy.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 3.3.0
- Chromium version 83.0.4103.0

Changes in syn-nodejs-2.0-beta:

- **Upgraded dependencies**— This runtime version uses Puppeteer-core version 3.3.0 and Chromium version 83.0.4103.0
- **Synthetics reporting**— For each canary run, CloudWatch Synthetics creates a report named `SyntheticsReport-PASSED.json` or `SyntheticsReport-FAILED.json` which records data such as start time, end time, status, and failures. It also records the PASSED/FAILED status of each step of the canary script, and failures and screenshots captured for each step.

- **Broken link checker report**— The new version of the broken link checker included in this runtime creates a report that includes the links that were checked, status code, failure reason (if any), and source and destination page screenshots.
- **New CloudWatch metrics**— Synthetics publishes metrics named `2xx`, `4xx`, `5xx`, and `RequestFailed` in the `CloudWatchSynthetics` namespace. These metrics show the number of 200s, 400s, 500s, and request failures in the canary runs. These metrics are reported only for UI canaries, and are not reported for API canaries.
- **Sortable HAR files**— You can now sort your HAR files by status code, request size, and duration.
- **Metrics timestamp**— CloudWatch metrics are now reported based on the Lambda invocation time instead of the canary run end time.

Bug fixes in syn-nodejs-2.0-beta:

- Fixed the issue of canary artifact upload errors not being reported. Such errors are now surfaced as execution errors.
- Fixed the issue of redirected requests (3xx) being incorrectly logged as errors.
- Fixed the issue of screenshots being numbered starting from 0. They should now start with 1.
- Fixed the issue of screenshots being garbled for Chinese and Japanese fonts.

syn-1.0

The first Synthetics runtime version is `syn-1.0`.

Major dependencies:

- Lambda runtime Node.js 10.x
- Puppeteer-core version 1.14.0
- The Chromium version that matches Puppeteer-core 1.14.0

Runtime versions using Python and Selenium Webdriver

The following sections contain information about the CloudWatch Synthetics runtime versions for Python and Selenium Webdriver. Selenium is an open-source browser automation tool. For more information about Selenium, see www.selenium.dev/

For features and methods supported by Synthetics runtime on Selenium framework, see Python and Selenium library classes and functions that apply to UI canaries only .

The naming convention for these runtime versions is `syn-`*`language`*`-`*`framework`*`-`*`majorversion`*`.` *`minorversion`*.

syn-python-selenium-13.0

Major dependencies:

- Python 3.12
- Selenium 4.32.0
- Chromium version 153.0.8010.36

Changes in syn-python-selenium-13.0

- Upgrade `Chromium` to 153.0.8010.36 to address the following CVEs:
  - CVE-2026-85046
  - CVE-2026-87491

For more information, see the following:

- Selenium changelog on the Selenium website
- Selenium API documentation on the Selenium website

Previous runtime versions for Python and Selenium

The following earlier runtime versions for Python and Selenium are still supported.

syn-python-selenium-12.0

Major dependencies:

- Python 3.12
- Selenium 4.32.0
- Chromium version 150.0.7871.24

Changes in syn-python-selenium-12.0

- Upgrade `Chromium` to 150.0.7871.24 to address the following CVEs:
  - CVE-2026-11645

syn-python-selenium-11.1

Major dependencies:

- Python 3.12
- Selenium 4.32.0
- Chromium version 147.0.7727.57

Changes in syn-python-selenium-11.1

- Upgrade `urllib3` to 2.7.0 to address the following CVEs:
  - CVE-2026-44431
  - CVE-2026-44432

syn-python-selenium-11.0

Major dependencies:

- Python 3.12
- Selenium 4.32.0
- Chromium version 147.0.7727.57

Changes in syn-python-selenium-11.0

- Updated Python runtime and browser versions.
- Upgrade `Chromium` to 147.0.7727.57 to address the following CVEs:
  - CVE-2026-3909
  - CVE-2026-3910
  - CVE-2026-5281

syn-python-selenium-10.0

Major dependencies:

- Python 3.11
- Selenium 4.32.0
- Chromium version 145.0.7632.77

Changes in syn-python-selenium-10.0

- Applied security patches and updated browser versions.

syn-python-selenium-9.0

Major dependencies:

- Python 3.11
- Selenium 4.32.0
- Chromium version 143.0.7499.169

Changes in syn-python-selenium-9.0

- Applied security patches and updated browser versions.

syn-python-selenium-8.0

Major dependencies:

- Python 3.11
- Selenium 4.32.0
- Chromium version 142.0.7444.175

Changes in syn-python-selenium-8.0

- Applied security patches and updated Selenium and browser versions.
- Modified failed HAR network request log level from ERROR to INFO.

syn-python-selenium-7.0

Major dependencies:

- Python 3.11
- Selenium 4.32.0
- Chromium version 138.0.7204.168

Changes in syn-python-selenium-7.0

- Applied security patches and updated Selenium and browser versions.

syn-python-selenium-6.0

Major dependencies:

- Python 3.11
- Selenium 4.21.0
- Chromium version 131.0.6778.264

Changes in syn-python-selenium-6.0

- Upgrade from Python 3.9 to Python 3.11.

syn-python-selenium-5.1

Major dependencies:

- Python 3.9
- Selenium 4.21.0
- Chromium version 131.0.6778.264

Changes in syn-python-selenium-5.1

- Minor updates on metric emission.
- Supports dry runs for the canary which allows for adhoc executions or performing a safe canary update.

syn-python-selenium-5.0

Major dependencies:

- Python 3.9
- Selenium 4.21.0
- Chromium version 131.0.6778.264

Changes in syn-python-selenium-5.0:

- Automatic retry if the browser fails to launch.

syn-python-selenium-4.1

Major dependencies:

- Python 3.9
- Selenium 4.15.1
- Chromium version 126.0.6478.126

Changes in syn-python-selenium-4.1:

- **Addresses security vulnerability**– This runtime has an update to address the CVE-2024-39689 vulnerability.

syn-python-selenium-4.0

Major dependencies:

- Python 3.9
- Selenium 4.15.1
- Chromium version 126.0.6478.126

Changes in syn-python-selenium-4.0:

- **Bug fixes** for errors in HAR parser logging.

Deprecated runtime versions for Python and Selenium

The following earlier runtime versions for Python and Selenium have been deprecated. For information about runtime deprecation dates, see CloudWatch Synthetics runtime deprecation dates.

syn-python-selenium-3.0

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 121.0.6167.139

Changes in syn-python-selenium-3.0:

- **Updated versions of the bundled libraries in Chromium**— The Chromium dependency is updated to a new version.

syn-python-selenium-2.1

Major dependencies:

- Python 3.8
- Selenium 4.15.1
- Chromium version 111.0.5563.146

Changes in syn-python-selenium-2.1:

- **Updated versions of the bundled libraries in Chromium**— The Chromium and Selenium dependencies are updated to new versions.

syn-python-selenium-2.0

Major dependencies:

- Python 3.8

- Selenium 4.10.0
- Chromium version 111.0.5563.146

Changes in syn-python-selenium-2.0:

- **Updated dependencies**— The Chromium and Selenium dependencies are updated to new versions.

Bug fixes in syn-python-selenium-2.0:

- **Timestamp added**— A timestamp has been added to canary logs.
- **Session re-use**— A bug was fixed so that canaries are now prevented from reusing the session from their previous canary run.

syn-python-selenium-1.3

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

Changes in syn-python-selenium-1.3:

- **More precise timestamps**— The start time and stop time of canary runs are now precise to the millisecond.

syn-python-selenium-1.2

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 92.0.4512.0

- **Updated dependencies**— The only new features in this runtime are the updated dependencies.

syn-python-selenium-1.1

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Custom handler function**— You can now use a custom handler function for your canary scripts. Previous runtimes required the script entry point to include `.handler`.

  You can also put canary scripts in any folder and pass the folder name as part of the handler. For example, `MyFolder/MyScriptFile.functionname` can be used as an entry point.
- **Configuration options for adding metrics and step failure configurations**— These options were already available in runtimes for Node.js canaries. For more information, see SyntheticsConfiguration class .
- **Custom arguments in Chrome** — You can now open a browser in incognito mode or pass in proxy server configuration. For more information, see Chrome().
- **Cross-Region artifact buckets**— A canary can store its artifacts in an Amazon S3 bucket in a different Region.
- **Bug fixes, including a fix for the `index.py` issue**— With previous runtimes, a canary file named `index.py` caused exceptions because it conflicted with the name of the library file. This issue is now fixed.

syn-python-selenium-1.0

Major dependencies:

- Python 3.8
- Selenium 3.141.0
- Chromium version 83.0.4103.0

Features:

- **Selenium support**— You can write canary scripts using the Selenium test framework. You can bring your Selenium scripts from elsewhere into CloudWatch Synthetics with minimal changes, and they will work with AWS services.

Runtime versions using Node.js

The following section contains information about the CloudWatch Synthetics runtime versions for Node.js. This runtime does not have any browser or framework included.

The naming convention for these runtime versions is `syn-`*`language`*`-`*`majorversion`*`.`*`minorversion`*.

> **Important**
>
> Starting Synthetics `syn-nodejs-3.1` and later, Synthetics runtime uses the new namespace. Please migrate the canary script to use the new namespace. Legacy namespace will be deprecated in a future release.
>
> - @amzn/synthetics-core → @aws/synthetics-core

syn-nodejs-5.3

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-5.3

- Multi-check canary dependency fix – Fix a dependency conflict that could prevent multi-check canaries from starting.
- Upgrade `fast-uri` to 3.1.8 to address the following CVEs:
  - CVE-2026-16221
  - CVE-2026-18446
  - CVE-2026-75899
  - CVE-2026-75931
  - CVE-2026-75975

- CVE-2026-76172
- Upgrade `protobufjs` to 7.6.6 to address the following CVE:
  - CVE-2026-59877
- Upgrade `axios` to 1.18.0 to address the following CVEs:
  - CVE-2026-67312
  - CVE-2026-67313
  - CVE-2026-67314
  - CVE-2026-67315
  - CVE-2026-67316
  - CVE-2026-67317
  - CVE-2026-67318
  - CVE-2026-67319
  - CVE-2026-67320
  - CVE-2026-67321

Previous runtime versions for Node.js

The following earlier runtime versions for Node.js are still supported.

syn-nodejs-5.2

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-5.2

- Multi checks blueprint bug fix – The runtime now correctly resolves `${AWS_SECRET:...}` references in global variables and SigV4 authentication configuration to their stored secret values.
- Upgrade `protobufjs` to 7.5.6 to address the following CVEs:
  - CVE-2026-41242
- Upgrade `jsonpath` to 1.3.0 to address the following CVEs:
  - CVE-2026-1615

- Upgrade `fast-xml-parser` to 5.9.3 to address the following CVEs:
  - CVE-2026-25896

syn-nodejs-5.1

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-5.1

- Fix bug where HTTP headers with array values were not being captured properly.
- Upgrade `ws` to 8.20.1 to address the following CVEs:
  - CVE-2026-45736

syn-nodejs-5.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-5.0

- Upgrade `brace-expansion` to 5.0.6 to address the following CVE:
  - GHSA-jxxr-4gwj-5jf2

syn-nodejs-4.2

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-4.2

- Upgrade `axios` to 1.15.2 to address the following CVEs:
  - CVE-2025-62718
  - CVE-2026-42033

- CVE-2026-42035
- CVE-2026-42038
- CVE-2026-42039
- CVE-2026-42043
- CVE-2026-42044
- CVE-2026-42264

- Upgrade `brace-expansion` to 5.0.5 to address the following CVE:
  - CVE-2026-33750
- Upgrade `fast-xml-builder` to 1.1.7 to address the following CVE:
  - GHSA-5wm8-gmm8-39j9

syn-nodejs-4.1

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-4.1

- Upgrade `fast-xml-parser` to 5.5.7 to address the following CVEs:
  - CVE-2026-25128
  - CVE-2026-25896
  - CVE-2026-26278
  - CVE-2026-27942
  - CVE-2026-33036

syn-nodejs-4.0

Major dependencies:

- AWS Lambda runtime Node.js 22.x

Changes in syn-nodejs-4.0

- Applied security patches.

syn-nodejs-3.1

Major dependencies:

- AWS Lambda runtime Node.js 20.x

Changes in syn-nodejs-3.1

- Synthetics runtime namespace migration.
- Type definition is available in npm Registry. Please make sure the type definition package version matches your canary's runtime version.

syn-nodejs-3.0

Major dependencies:

- AWS Lambda runtime Node.js 20.x

Changes in syn-nodejs-3.0

- Support for multi checks blueprint.

Runtime versions support policy

Synthetics runtime versions are subject to maintenance and security updates. When any component of a runtime version is no longer supported, that Synthetics runtime version is deprecated.

You can't create canaries using deprecated runtime versions. Canaries that use deprecated runtimes continue to run. You can stop, start, and delete these canaries. You can update an existing canary that uses a deprecated runtime version by updating the canary to use a supported runtime version.

CloudWatch Synthetics notifies you by email if you have canaries that use runtimes that are scheduled to be deprecated in the next 60 days. We recommend that you migrate your canaries to a supported runtime version to benefit from the new functionality, security, and performance enhancements that are included in more recent releases.

CloudWatch Synthetics runtime deprecation dates

The following table lists the date of deprecation of each deprecated CloudWatch Synthetics runtime.

Runtime Version	Deprecation date
`syn-python-selenium-5.1`	February 3, 2026
`syn-python-selenium-5.0`	February 3, 2026
`syn-python-selenium-4.1`	February 3, 2026
`syn-python-selenium-4.0`	February 3, 2026
`syn-nodejs-puppeteer-7.0`	January 22, 2026
`syn-nodejs-puppeteer-6.2`	January 22, 2026
`syn-nodejs-puppeteer-5.2`	January 22, 2026
`syn-python-selenium-3.0`	January 22, 2026
`syn-python-selenium-2.1`	January 22, 2026
`syn-nodejs-puppeteer-6.1`	March 8, 2024
`syn-nodejs-puppeteer-6.0`	March 8, 2024

Runtime Version	Deprecation date
syn-nodejs-puppeteer-5.1	March 8, 2024
syn-nodejs-puppeteer-5.0	March 8, 2024
syn-nodejs-puppeteer-4.0	March 8, 2024
syn-nodejs-puppeteer-3.9	January 8, 2024
syn-nodejs-puppeteer-3.8	January 8, 2024
syn-python-selenium-2.0	March 8, 2024
syn-python-selenium-1.3	March 8, 2024
syn-python-selenium-1.2	March 8, 2024
syn-python-selenium-1.1	March 8, 2024
syn-python-selenium-1.0	March 8, 2024
syn-nodejs-puppeteer-3.7	January 8, 2024
syn-nodejs-puppeteer-3.6	January 8, 2024

Runtime Version	Deprecation date
`syn-nodejs-puppeteer-3.5`	January 8, 2024
`syn-nodejs-puppeteer-3.4`	November 13, 2022
`syn-nodejs-puppeteer-3.3`	November 13, 2022
`syn-nodejs-puppeteer-3.2`	November 13, 2022
`syn-nodejs-puppeteer-3.1`	November 13, 2022
`syn-nodejs-puppeteer-3.0`	November 13, 2022
`syn-nodejs-2.2`	May 28, 2021
`syn-nodejs-2.1`	May 28, 2021
`syn-nodejs-2.0`	May 28, 2021
`syn-nodejs-2.0-beta`	February 8, 2021
`syn-1.0`	May 28, 2021

Runtime versions update

You can update a canary's runtime version by using the CloudWatch console, AWS CloudFormation, the AWS CLI or the AWS SDK. When you use the CloudWatch console, you can update up to five canaries at once by selecting them in the canary list page and then choosing **Actions** , **Update Runtime**.

You can verify the update by testing it first before committing the runtime update. When updating the runtime versions, choose the **Start Dry Run** or **Validate and save later** options in the

CloudWatch console to create a dry run of the original canary along with any changes you made to the configuration. The dry run will update and execute the canary to validate whether the runtime update is safe for the canary. After you have verified your canary with the new runtime version, you can update the runtime version of your canary. For more information, see Performing safe canary updates.

Alternatively, you can verify the update by first cloning the canary using the CloudWatch console and updating the runtime version. This creates another canary which is a clone of your original canary. After you have verified your canary with the new runtime version, you can update the runtime version of your original canary and delete the clone canary.

You can also update multiple canaries using an upgrade script. For more information, see Canary runtime upgrade script.

If you upgrade a canary and it fails, see Troubleshooting a failed canary.

Canary runtime upgrade script

To upgrade a canary script to a supported runtime version, use the following script.

```
const AWS = require('aws-sdk');

// You need to configure your AWS credentials and Region.
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-
credentials-node.html
//   https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/setting-
region.html

const synthetics = new AWS.Synthetics();

const DEFAULT_OPTIONS = {
  /**
   * The number of canaries to upgrade during a single run of this script.
   */
  count: 10,
  /**
   * No canaries are upgraded unless force is specified.
   */
  force: false
};

/**
 * The number of milliseconds to sleep between GetCanary calls when
```

```
 * verifying that an update succeeded.
 */
const SLEEP_TIME = 5000;

(async () => {
  try {
    const options = getOptions();

    const versions = await getRuntimeVersions();
    const canaries = await getAllCanaries();
    const upgrades = canaries
      .filter(canary => !versions.isLatestVersion(canary.RuntimeVersion))
      .map(canary => {
        return {
          Name: canary.Name,
          FromVersion: canary.RuntimeVersion,
          ToVersion: versions.getLatestVersion(canary.RuntimeVersion)
        };
      });

    if (options.force) {
      const promises = [];

      for (const upgrade of upgrades.slice(0, options.count)) {
        const promise = upgradeCanary(upgrade);
        promises.push(promise);
        // Sleep for 100 milliseconds to avoid throttling.
        await usleep(100);
      }

      const succeeded = [];
      const failed = [];
      for (let i = 0; i < upgrades.slice(0, options.count).length; i++) {
        const upgrade = upgrades[i];
        const promise = promises[i];
        try {
          await promise;
          console.log(`The update of ${upgrade.Name} succeeded.`);
          succeeded.push(upgrade.Name);
        } catch (e) {
          console.log(`The update of ${upgrade.Name} failed with error: ${e}`);
          failed.push({
            Name: upgrade.Name,
            Reason: e
```

```
          });
        }
      }

      if (succeeded.length) {
        console.group('The following canaries were upgraded successfully.');
        for (const name of succeeded) {
          console.log(name);
        }
        console.groupEnd()
      } else {
        console.log('No canaries were upgraded successfully.');
      }

      if (failed.length) {
        console.group('The following canaries were not upgraded successfully.');
        for (const failure of failed) {
          console.log('\x1b[31m', `${failure.Name}: ${failure.Reason}`, '\x1b[0m');
        }
        console.groupEnd();
      }
    } else {
      console.log('Run with --force [--count <count>] to perform the first <count>
 upgrades shown. The default value of <count> is 10.')
      console.table(upgrades);
    }
  } catch (e) {
    console.error(e);
  }
})();

function getOptions() {
  const force = getFlag('--force', DEFAULT_OPTIONS.force);
  const count = getOption('--count', DEFAULT_OPTIONS.count);
  return { force, count };

  function getFlag(key, defaultValue) {
    return process.argv.includes(key) || defaultValue;
  }
  function getOption(key, defaultValue) {
    const index = process.argv.indexOf(key);
    if (index < 0) {
      return defaultValue;
    }
```

```
    const value = process.argv[index + 1];
    if (typeof value === 'undefined' || value.startsWith('-')) {
      throw `The ${key} option requires a value.`;
    }
    return value;
  }
}

function getAllCanaries() {
  return new Promise((resolve, reject) => {
    const canaries = [];

    synthetics.describeCanaries().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          resolve(canaries);
        } else {
          canaries.push(...data.Canaries);
        }
      }
    });
  });
}

function getRuntimeVersions() {
  return new Promise((resolve, reject) => {
    const jsVersions = [];
    const pythonVersions = [];
    synthetics.describeRuntimeVersions().eachPage((err, data) => {
      if (err) {
        reject(err);
      } else {
        if (data === null) {
          jsVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          pythonVersions.sort((a, b) => a.ReleaseDate - b.ReleaseDate);
          resolve({
            isLatestVersion(version) {
              const latest = this.getLatestVersion(version);
              return latest === version;
            },
            getLatestVersion(version) {
              if (jsVersions.some(v => v.VersionName === version)) {
```

```
                return jsVersions[jsVersions.length - 1].VersionName;
              } else if (pythonVersions.some(v => v.VersionName === version)) {
                return pythonVersions[pythonVersions.length - 1].VersionName;
              } else {
                throw Error(`Unknown version ${version}`);
              }
            }
          });
        } else {
          for (const version of data.RuntimeVersions) {
            if (version.VersionName === 'syn-1.0') {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-2.')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-nodejs-puppeteer-')) {
              jsVersions.push(version);
            } else if (version.VersionName.startsWith('syn-python-selenium-')) {
              pythonVersions.push(version);
            } else {
              throw Error(`Unknown version ${version.VersionName}`);
            }
          }
        }
      }
    });
  });
}

async function upgradeCanary(upgrade) {
  console.log(`Upgrading canary ${upgrade.Name} from ${upgrade.FromVersion} to
 ${upgrade.ToVersion}`);
  await synthetics.updateCanary({ Name: upgrade.Name, RuntimeVersion:
 upgrade.ToVersion }).promise();
  while (true) {
    await usleep(SLEEP_TIME);
    console.log(`Getting the state of canary ${upgrade.Name}`);
    const response = await synthetics.getCanary({ Name: upgrade.Name }).promise();
    const state = response.Canary.Status.State;
    console.log(`The state of canary ${upgrade.Name} is ${state}`);
    if (state === 'ERROR' || response.Canary.Status.StateReason) {
      throw response.Canary.Status.StateReason;
    }
    if (state !== 'UPDATING') {
      return;
```

```
    }
  }
}

function usleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

Writing a canary script

The following sections explain how to write a canary script and how to integrate a canary with other AWS services and with external dependencies and libraries.

Topics

- Common features for CloudWatch Synthetics canaries
- Writing a canary script using the Java runtime
- Writing a Node.js canary script using the Playwright runtime
- Writing a Node.js canary script using the Puppeteer runtime
- Writing a Python canary script
- Writing a JSON configuration for Node.js multi Checks blueprint

Common features for CloudWatch Synthetics canaries

You can use the following features with all canary runtimes.

Environment variables

You can use environment variables when you create canaries. You can write a single canary script and use it with different values to quickly create multiple canaries for a similar task.

For example, suppose your organization has endpoints such as `prod`, `dev`, and `pre-release` for different software development stages. You must create canaries to test each endpoint. You can write a single canary script that tests your software. Then specify different endpoint environment variable values when you create each of the three canaries. When you create a canary, specify the script and the environment variable values.

The names of environment variables can contain letters, numbers, and the underscore character. They must start with a letter and be at least two characters. The total size of your environment

variables can't exceed 4 KB. You can't specify any Lambda reserved environment variables as the names of your environment variables. For more information about reserved environment variables, see Runtime environment variables.

> **⚠ Environment variables are not encrypted client-side**
>
> By default, AWS encrypts environment variable keys and values at rest using an AWS owned key. However, CloudWatch Synthetics does not apply client-side encryption. Store sensitive information only after encrypting it in transit. For more information, see Encrypting environment variables in transit. You can also use a customer managed AWS KMS key to encrypt your canary's environment variables at rest. For more information, see Encrypting environment variables at rest with a customer managed key.

The following example script uses two environment variables. This script is for a canary that checks whether a webpage is available. It uses environment variables to parameterize both the URL that it checks and the CloudWatch Synthetics log level that it uses.

The following snippets are part of the complete script that follows.

The following function sets `LogLevel` to the value of the `LOG_LEVEL` environment variable.

```
synthetics.setLogLevel(process.env.LOG_LEVEL);
```

This function sets `URL` to the value of the `URL` environment variable.

```
const URL = process.env.URL;
```

The following complete script demonstrates both environment variables. When you create a canary using this script, you specify values for the `LOG_LEVEL` and `URL` environment variables.

```
var synthetics = require('@aws/synthetics-puppeteer');
const log = require('@aws/synthetics-logger');

const pageLoadEnvironmentVariable = async function () {

  // Setting the log level (0-3)
  synthetics.setLogLevel(process.env.LOG_LEVEL);
  // INSERT URL here
```

```
  const URL = process.env.URL;

  let page = await synthetics.getPage();
  //You can customize the wait condition here. For instance,
  //using 'networkidle2' may be less restrictive.
  const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
  if (!response) {
      throw "Failed to load page!";
  }
  //Wait for page to render.
  //Increase or decrease wait time based on endpoint being monitored.
  await page.waitFor(15000);
  await synthetics.takeScreenshot('loaded', 'loaded');
  let pageTitle = await page.title();
  log.info('Page title: ' + pageTitle);
  log.debug('Environment variable:' + process.env.URL);

  //If the response status code is not a 2xx success code
  if (response.status() < 200 || response.status() > 299) {
      throw "Failed to load page!";
  }
};

exports.handler = async () => {
  return await pageLoadEnvironmentVariable();
};
```

Passing environment variables to your script

To pass environment variables to your script when you create a canary in the console, specify the keys and values of the environment variables in the **Environment variables** section on the console. For more information, see Creating a canary.

To pass environment variables through the API or AWS CLI, use the `EnvironmentVariables` parameter in the `RunConfig` section. The following is an example AWS CLI command that creates a canary that uses two environment variables with keys of `Environment` and `Region`.

For the full schema of the `RunConfig` parameter including `EnvironmentVariables`, see CanaryRunConfigInput in the *Amazon CloudWatch Synthetics API Reference*.

```
aws synthetics create-canary --cli-input-json '{
  "Name": "nameofCanary",
```

```
  "ExecutionRoleArn": "roleArn",
  "ArtifactS3Location": "s3://amzn-s3-demo-bucket-123456789012-us-west-2",
  "Schedule": {
    "Expression": "rate(0 minute)",
    "DurationInSeconds": 604800
  },
  "Code": {
    "S3Bucket": "canarycreation",
    "S3Key": "cwsyn-mycanaryheartbeat-12345678-d1bd-1234-
abcd-123456789012-12345678-6a1f-47c3-b291-123456789012.zip",
    "Handler": "pageLoadBlueprint.handler"
  },
  "RunConfig": {
    "TimeoutInSeconds": 60,
    "EnvironmentVariables": {
      "Environment": "Production",
      "Region": "us-west-1"
    }
  },
  "SuccessRetentionPeriodInDays": 13,
  "FailureRetentionPeriodInDays": 13,
  "RuntimeVersion": "syn-nodejs-2.0"
}'
```

Encrypting environment variables at rest with a customer managed key

By default, an AWS owned key encrypts canary environment variables at rest. You can specify a customer managed AWS KMS key to encrypt the canary environment variables at rest. With a customer managed key, you have full control over the encryption of sensitive configuration data. The following sections describe the requirements, configuration steps, and permissions for using a customer managed key.

Requirements

Before you configure a customer managed key, verify that you meet the following requirements:

- The AWS KMS key must be a symmetric encryption key.
- The key policy must grant `kms:CreateGrant` to the caller (the IAM principal calling the Synthetics API).
- AWS Lambda uses the grant to encrypt and decrypt the environment variables at rest.
- The AWS KMS key must be in the same AWS Region as the canary.

Configuring a customer managed key

You can configure a customer managed key when you create or update a canary. The following procedures show how to configure encryption using the Amazon CloudWatch console and the Synthetics API.

Configure encryption in the console

To create or edit a canary in the Amazon CloudWatch console, expand the **Environment variables** section. Under **Encryption at rest configuration**, choose **Use a customer managed key** and then choose or specify the ARN of your AWS KMS key.

Configure encryption using the API

When calling `CreateCanary` or `UpdateCanary`, specify the `KmsKeyArn` parameter with the ARN of your customer managed key. To revert to the AWS managed key, set `KmsKeyArn` to an empty string.

Example: CreateCanary request with a customer managed key

```
{
  "Name": "my-canary-EXAMPLE",
  "KmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/a1b2c3d4-e5f6-7890-abcd-
EXAMPLE11111",
  "RunConfig": {
    "EnvironmentVariables": {
      "SECRET_KEY": "my-secret-value-EXAMPLE"
    }
  }
}
```

Required permissions for at-rest encryption

If you create or update the canary, you must have the following permissions on the AWS KMS key:

- `kms:CreateGrant`, `kms:Encrypt`—Required to configure a customer managed key for the canary.
- `kms:Decrypt`—Required to view and manage environment variables that are encrypted with a customer managed key.
- `kms:DescribeKey`—Required to validate the key.

The canary execution role requires no AWS KMS permissions for at-rest encryption. Lambda uses the grant to handle encryption and decryption.

Multi-location canaries

For multi-location canaries, each replica location can have its own AWS KMS key. Specify the `KmsKeyArn` in the `AddReplicaLocations` parameter when creating or updating a canary. The key must be in the same Region as the replica.

Encrypting environment variables in transit

In addition to encryption at rest, you can encrypt individual environment variable values before CloudWatch Synthetics stores them. CloudWatch Synthetics calls this encryption in transit. When you encrypt a value in transit, the console replaces the plaintext value with a base64-encoded ciphertext that only your canary can decrypt at runtime.

How encryption in transit works

When you choose to encrypt an environment variable value in transit:

1. The console calls `kms:Encrypt` with your selected AWS KMS key to encrypt the plaintext value.
2. The encrypted ciphertext (base64-encoded) replaces the plaintext value in the environment variable configuration.
3. At runtime, your canary script decrypts the value by calling `kms:Decrypt`.

Required permissions for encryption in transit

You need the following permissions for encryption in transit:

- **Console user or API caller**— `kms:Encrypt` on the AWS KMS key. You need this permission to encrypt the value before storing it.
- **Canary execution role**— `kms:Decrypt` on the AWS KMS key. The canary's Lambda function needs this permission to decrypt the value at runtime.

The following is an example IAM policy to attach to the canary execution role:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:us-east-1:111122223333:key/a1b2c3d4-e5f6-7890-abcd-
EXAMPLE11111"
    }
  ]
}
```

Decrypting values in your canary script

To use an encrypted environment variable in your canary script, decrypt it at runtime. The following Node.js example shows how to decrypt an environment variable:

```
const { KMSClient, DecryptCommand } = require('@aws-sdk/client-kms');
const client = new KMSClient({ region: process.env.AWS_REGION });

async function decryptEnvVar(name) {
  const encrypted = process.env[name];
  const req = {
    CiphertextBlob: Buffer.from(encrypted, 'base64'),
  };
  const command = new DecryptCommand(req);
  const response = await client.send(command);
  return new TextDecoder().decode(response.Plaintext);
}

// Usage
const mySecret = await decryptEnvVar('MY_CONFIG_VAR');
```

Integrating your canary with other AWS services

You can use the AWS SDK library in your canary to integrate with other AWS services.

To do so, add the following code to your canary. In these examples, the canary integrates with AWS Secrets Manager.

- Import the AWS SDK.

  ```
  const AWS = require('aws-sdk');
  ```

- Create a client for the AWS service that you are integrating with.

```
const secretsManager = new AWS.SecretsManager();
```

- Use the client to make API calls to that service.

```
var params = {
SecretId: secretName
};
return await secretsManager.getSecretValue(params).promise();
```

The following canary script code snippet shows how to integrate with Secrets Manager in more detail.

```
var synthetics = require('@aws/synthetics-puppeteer');
const log = require('@aws/synthetics-logger');

const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager();

const getSecrets = async (secretName) => {
  var params = {
      SecretId: secretName
  };
  return await secretsManager.getSecretValue(params).promise();
}

const secretsExample = async function () {
  let URL = "<URL>";
  let page = await synthetics.getPage();

  log.info(`Navigating to URL: ${URL}`);
  const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});

  // Fetch secrets
  let secrets = await getSecrets("secretname")

  /**
  * Use secrets to login.
  *
  * Assuming secrets are stored in a JSON format like:
  * {
```

```
  *   "username": "<USERNAME>",
  *   "password": "<PASSWORD>"
  * }
  **/
  let secretsObj = JSON.parse(secrets.SecretString);
  await synthetics.executeStep('login', async function () {
      await page.type(">USERNAME-INPUT-SELECTOR<", secretsObj.username);
      await page.type(">PASSWORD-INPUT-SELECTOR<", secretsObj.password);

      await Promise.all([
        page.waitForNavigation({ timeout: 30000 }),
        await page.click(">SUBMIT-BUTTON-SELECTOR<")
      ]);
  });

  // Verify login was successful
  await synthetics.executeStep('verify', async function () {
      await page.waitForXPath(">SELECTOR<", { timeout: 30000 });
  });
};

exports.handler = async () => {
  return await secretsExample();
};
```

Forcing your canary to use a static IP address

You can set up a canary so that it uses a static IP address.

To force a canary to use a static IP address

1. Create a new VPC. For more information, see Using DNS with your VPC.
2. Create a new internet gateway. For more information, see Adding an internet gateway to your VPC.
3. Create a public subnet inside your new VPC.
4. Add a new route table to the VPC.
5. Add a route in the new route table that goes from `0.0.0.0/0` to the internet gateway.
6. Associate the new route table with the public subnet.
7. Create an elastic IP address. For more information, see Elastic IP addresses.
8. Create a new NAT gateway and assign it to the public subnet and the elastic IP address.

9. Create a private subnet inside the VPC.
10. Add a route to the VPC default route table that goes from `0.0.0.0/0` to the NAT gateway.
11. Create your canary.

Writing a canary script using the Java runtime

Topics

- Java project structure for a canary
- Packaging the project for a canary
- Handler name
- CloudWatch Synthetics configurations
- CloudWatch Synthetics environment variables

Java project structure for a canary

To create a canary in Java, you need to write your code, compile it, and deploy the compiled artifacts to Synthetics. You can initialize a Java Lambda project in various ways. For instance, you can use a standard Java project setup in your preferred IDE, such as IntelliJ IDEA or Visual Studio Code. Alternatively, you can create the required file structure manually.

A Synthetics Java project contains the following general structure:

```
/project-root
    # src
        # main
            # java
                # canarypackage // name of package
                |    # ExampleCanary.java // Canary code file
                |    # other_supporting_classes
                - resources
                     # synthetics.json // Synthetics configuration file
     # build.gradle OR pom.xml
```

You can use either Maven or Gradle to build your project and manage dependencies.

In the above structure, the `ExampleCanary` class is the entry point or handler for the canary.

Java canary class example

This example is for a canary to make a get request to a URL stored in the *TESTING_URL* Lambda environment variable. The canary does not use any of the methods provided by Synthetics runtime.

```
package canarypackage;

import java.net.HttpURLConnection;
import java.net.URL;

// Handler value: canary.ExampleCanary::canaryCode
public class ExampleCanary {
  public void canaryCode() throws Exception{
      URL url = new URL(System.getenv("TESTING_URL"));
      HttpURLConnection con=(HttpURLConnection)url.openConnection();
      con.setRequestMethod("GET");
      con.setConnectTimeout(5000);
      con.setReadTimeout(5000);
      int status=con.getResponseCode();
      if(status!=200){
        throw new Exception("Failed to load " + url + ", status code: " + status);
      }
  }
}
```

It is highly recommended to modularize your canaries using the Synthetics provided library function `executeStep`. The canary makes `get` calls to two separate URLs procured from URL1 and URL2 environment variables.

> **Note**
>
> To use the `executeStep` functionality, the handler method for the canary should take in a parameter of type Synthetics as shown below.

```
package canarypackage;

import com.amazonaws.synthetics.Synthetics;
import java.net.HttpURLConnection;
import java.net.URL;

// Handler value: canary.ExampleCanary::canaryCode
public class ExampleCanary {
```

```
  public void canaryCode(Synthetics synthetics) throws Exception {
    createStep("Step1", synthetics, System.getenv("URL1"));
    createStep("Step2", synthetics, System.getenv("URL2"));
    return;
  }

  private void createStep(String stepName, Synthetics synthetics, String url) throws
 Exception{
    synthetics.executeStep(stepName,()->{
      URL obj=new URL(url);
      HttpURLConnection con=(HttpURLConnection)obj.openConnection();
      con.setRequestMethod("GET");
      con.setConnectTimeout(5000);
      con.setReadTimeout(5000);
      int status=con.getResponseCode();
      if(status!=200){
        throw new Exception("Failed to load" + url + "status code:" + status);
      }
      return null;
    }).get();
  }
}
```

Packaging the project for a canary

Synthetics accepts code for a java canary in the *zip* format. The zip consists of the class files for the canary code, the jars for any third party dependencies and the Synthetics configuration file.

A Synthetics Java zip contains the following general structure.

```
example-canary
    # lib
    |  # //third party dependency jars
       # java-canary.jar
    # synthetics.json
```

To build this zip from the above project structure, you can use gradle (build.gradle) or maven (pom.xml). Here is an example.

For information on compile time dependencies or interfaces for the Synthetics library, see the README under aws-cloudwatch-synthetics-sdk-java .

```
plugins {
```

```
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    // Third party dependencies
    // example: implementation 'software.amazon.awssdk:s3:2.31.9'

    // Declares dependency on Synthetics interfaces for compiling only
    // Refer https://github.com/aws/aws-cloudwatch-synthetics-sdk-java for building
 from source.
    compileOnly 'software.amazon.synthetics:aws-cloudwatch-synthetics-sdk-java:1.0.0'}

test {
    useJUnitPlatform()
}

// Build the zip to be used as Canary code.
task buildZip(type: Zip) {

    archiveFileName.set("example-canary.zip")
    destinationDirectory.set(file("$buildDir"))

    from processResources
    into('lib') {
        from configurations.runtimeClasspath
        from(tasks.named("jar"))
    }
    from "src/main/java/resources/synthetics.json"

    doLast {
        println "Artifact written to: ${archiveFile.get().asFile.absolutePath}"
    }
}

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}
```

```
tasks.named("build") {
    dependsOn "buildZip"
}
```

Handler name

The handler name is the entry point for the canary. For Java runtime, the handler is in the following format.

```
<<full qualified name for canary class>>::<<name of the method to start the execution
 from>>
// for above code: canarypackage.ExampleCanary::canaryCode
```

CloudWatch Synthetics configurations

You can configure the behavior of the Synthetics Java runtime by providing an optional JSON configuration file named `synthetics.json`. This file should be packaged in the root directory of the package zip. Though a configuration file is optional, if you don't provide a configuration file, or a configuration key is missing, CloudWatch uses the defaults.

The following are supported configuration values, and their defaults.

```
{
    "step": {
        "stepSuccessMetric": true,
        "stepDurationMetric": true,
        "continueOnStepFailure": false,
        "stepsReport": true
    },
    "logging": {
        "logRequest": false,
        "logResponse": false
    },
    "httpMetrics": {
        "metric_2xx": true,
        "metric_4xx": true,
        "metric_5xx": true,
        "aggregated2xxMetric": true,
        "aggregated4xxMetric": true,
        "aggregated5xxMetric": true
    },
```

```
    "canaryMetrics": {
        "failedCanaryMetric": true,
        "aggregatedFailedCanaryMetric": true
    }
}
```

Step configurations

- *continueOnStepFailure* – Determines if a script should continue even after a step has failed. The default is false.
- *stepSuccessMetric* – Determines if a step's `SuccessPercent` metric is emitted. The `SuccessPercent` metric for a step is *100* for the canary run if the step succeeds, and *0* if the step fails. The default is *true*.
- *stepDurationMetric* – Determines if a step's *Duration* metric is emitted. The *Duration* metric is emitted as a duration, in milliseconds, of the step's run. The default is *true*.

Logging configurations

Applies to logs generated by CloudWatch Synthetics. Controls the verbosity of request and response logs.

- *logRequest* – Specifies whether to log every request in canary logs. The default is false.
- *logResponse* – Specifies whether to log every response in canary logs. The default is false.

HTTP metric configurations

Configurations for metrics related to the count of network requests with different HTTP status codes, emitted by CloudWatch Synthetics for this canary.

- *metric_2xx* – Specifies whether to emit the *2xx* metric (with the CanaryName dimension) for this canary. The default is *true*.
- *metric_4xx* – Specifies whether to emit the *4xx* metric (with the CanaryName dimension) for this canary. The default is *true*.
- *metric_5xx* – Specifies whether to emit the *5xx* metric (with the CanaryName dimension) for this canary. The default is *true*.
- *aggregated2xxMetric* – Specifies whether to emit the *2xx* metric (without the CanaryName dimension) for this canary. The default is *true*.

- *aggregated4xxMetric* – Specifies whether to emit the *4xx* metric (without the CanaryName dimension) for this canary. The default is *true*.
- *aggregated5xxMetric* – Specifies whether to emit the *5xx* metric (without the CanaryName dimension) for this canary. The default is *true*.

Canary metric configurations

Configurations for other metrics emitted by CloudWatch Synthetics.

- *failedCanaryMetric* Network Access Analyzer Specifies whether to emit the *Failed* metric (with the CanaryName dimension) for this canary. The default is *true*.
- *aggregatedFailedCanaryMetric* – Specifies whether to emit the *Failed* metric (without the CanaryName dimension) for this canary. The default is *true*.

CloudWatch Synthetics environment variables

You can configure the logging level and format by using environment variables.

Log format

The CloudWatch Synthetics Java runtime creates CloudWatch logs for every canary run. Logs are written in JSON format for convenient querying. Optionally, you can change the log format to *TEXT*.

- *Environment variable name* – CW_SYNTHETICS_LOG_FORMAT
- *Supported values* – JSON, TEXT
- *Default* –JSON

Log levels

- *Environment variable name* – CW_SYNTHETICS_LOG_LEVEL
- *Supported values* – TRACE, DEBUG, INFO, WARN, ERROR, FATAL
- *Default* – INFO

Apart from the above environment variables, there is a default environment variable added for Java runtime, `AWS_LAMBDA-EXEC_WRAPPER` environment variable to your function, and set

its value to `/opt/synthetics-otel-instrument`. This environment variable modifies your function's startup behavior for telemetry. If this environment variable already exists, make sure that it's set to the required value.

Writing a Node.js canary script using the Playwright runtime

Topics

- Packaging your Node.js canary files for the Playwright runtime
- Changing an existing Playwright script to use as a CloudWatch Synthetics canary
- CloudWatch Synthetics configurations

Packaging your Node.js canary files for the Playwright runtime

Your canary script comprises of a `.js` (CommonJS syntax) or `.mjs` (ES syntax) file containing your Synthetics handler code, together with any additional packages and modules your code depends on. Scripts created in ES (ECMAScript) format should either use .mjs as the extension or include a package.json file with the "type": "module" field set. Unlike other runtimes like Node.js Puppeteer, you are not required to save your scripts in a specific folder structure. You can package your scripts directly. Use your preferred `zip` utility to create a `.zip` file with your handler file at the root. If your canary script depends on additional packages or modules that aren't included in the Synthetics runtime, you can add these dependencies to your `.zip` file. To do so, you can install your function's required libraries in the `node_modules` directory by running the `npm install` command. The following example CLI commands create a `.zip` file named `my_deployment_package.zip` containing the `index.js` or `index.mjs` file (Synthetics handler) and its dependencies. In the example, you install dependencies using the `npm` package manager.

```
~/my_function
### index.mjs
### synthetics.json
### myhelper-util.mjs
### node_modules
    ### mydependency
```

Create a `.zip` file that contains the contents of your project folder at the root. Use the `r` (recursive) option, as shown in the following example, to make sure that `zip` compresses the subfolders.

```
zip -r my_deployment_package.zip .
```

Add a Synthetics configuration file to configure the behavior of CloudWatch Synthetics. You can create a `synthetics.json` file and save it at the same path as your entry point or handler file.

Optionally, you can also store your entry point file in a folder structure of your choice. However, be sure that the folder path is specified in your handler name.

Handler name

Be sure to set your canary's script entry point (handler) as `myCanaryFilename.functionName` to match the file name of your script's entry point. You can optionally store the canary in a separate folder such as `myFolder/my_canary_filename.mjs`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Playwright script to use as a CloudWatch Synthetics canary

You can edit an existing script for Node.js and Playwright to be used as a canary. For more information about Playwright, see the Playwright library documentation.

You can use the following Playwright script that is saved in file `exampleCanary.mjs`.

```
import { chromium } from 'playwright';
import { expect } from '@playwright/test';

const browser = await chromium.launch();
const page = await browser.newPage();
await page.goto('https://example.com', {timeout: 30000});
await page.screenshot({path: 'example-home.png'});

const title = await page.title();
expect(title).toEqual("Example Domain");

await browser.close();
```

Convert the script by performing the following steps:

1. Create and export a `handler` function. The handler is the entry point function for the script. You can choose any name for the handler function, but the function that is used in your script should be the same as in your canary handler. If your script name is `exampleCanary.mjs`, and the handler function name is `myhandler`, your canary handler is named `exampleCanary.myhandler`. In the following example, the handler function name is `handler`.

```
exports.handler = async () => {
  // Your script here
  };
```

2. Import the `Synthetics Playwright module` as a dependency.

```
import { synthetics } from '@aws/synthetics-playwright';
```

3. Launch a browser using the Synthetics `Launch` function.

```
const browser = await synthetics.launch();
```

4. Create a new Playwright page by using the Synthetics `newPage` function.

```
const page = await synthetics.newPage();
```

Your script is now ready to be run as a Synthetics canary. The following is the the updated script:

Updated script in ES6 format

The script file saved with a `.mjs` extension.

```
import { synthetics } from '@aws/synthetics-playwright';
import { expect } from '@playwright/test';

export const handler = async (event, context) => {
  try {
        // Launch a browser
        const browser = await synthetics.launch();

        // Create a new page
        const page = await synthetics.newPage(browser);

        // Navigate to a website
        await page.goto('https://www.example.com', {timeout: 30000});

        // Take screenshot
        await page.screenshot({ path: '/tmp/example.png' });

        // Verify the page title
        const title = await page.title();
```

```
        expect(title).toEqual("Example Domain");
    } finally {
        // Ensure browser is closed
        await synthetics.close();
    }
};
```

Updated script in CommonJS format

The script file saved with a `.js` extension.

```
const { synthetics } = require('@aws/synthetics-playwright');
const { expect } = require('@playwright/test');

exports.handler = async (event) => {
  try {
    const browser = await synthetics.launch();
    const page = await synthetics.newPage(browser);
    await page.goto('https://www.example.com', {timeout: 30000});
    await page.screenshot({ path: '/tmp/example.png' });
    const title = await page.title();
    expect(title).toEqual("Example Domain");
  } finally {
    await synthetics.close();
  }
};
```

CloudWatch Synthetics configurations

You can configure the behavior of the Synthetics Playwright runtime by providing an optional JSON configuration file named `synthetics.json`. This file should be packaged in the same location as the handler file. Though a configuration file is optional, if you don't provide a configuration file, or a configuration key is missing, CloudWatch assumes defaults.

Packaging your configuration file

The following are supported configuration values, and their defaults.

```
{
    "step": {
        "screenshotOnStepStart": false,
        "screenshotOnStepSuccess": false,
        "screenshotOnStepFailure": false,
```

```
        "stepSuccessMetric": true,
        "stepDurationMetric": true,
        "continueOnStepFailure": true,
        "stepsReport": true
    },
    "report": {
        "includeRequestHeaders": true,
        "includeResponseHeaders": true,
        "includeUrlPassword": false,
        "includeRequestBody": true,
        "includeResponseBody": true,
        "restrictedHeaders": ['x-amz-security-token', 'Authorization'], // Value of
 these headers is redacted from logs and reports
        "restrictedUrlParameters": ['Session', 'SigninToken'] // Values of these url
 parameters are redacted from logs and reports
    },
    "logging": {
        "logRequest": false,
        "logResponse": false,
        "logResponseBody": false,
        "logRequestBody": false,
        "logRequestHeaders": false,
        "logResponseHeaders": false
    },
    "httpMetrics": {
        "metric_2xx": true,
        "metric_4xx": true,
        "metric_5xx": true,
        "failedRequestsMetric": true,
        "aggregatedFailedRequestsMetric": true,
        "aggregated2xxMetric": true,
        "aggregated4xxMetric": true,
        "aggregated5xxMetric": true
    },
    "canaryMetrics": {
        "failedCanaryMetric": true,
        "aggregatedFailedCanaryMetric": true
    },
    "userAgent": "",
    "har": true
}
```

Step configurations

- `screenshotOnStepStart` – Determines if Synthetics should capture a screenshot before the step starts. The default is `true`.
- `screenshotOnStepSuccess` – Determines if Synthetics should capture a screenshot after a step has succeeded. The default is `true`.
- `screenshotOnStepFailure` – Determines if Synthetics should capture a screenshot after a step has failed. The default is `true`.
- `continueOnStepFailure` – Determines if a script should continue even after a step has failed. The default is `false`.
- `stepSuccessMetric` – Determines if a step's `SuccessPercent` metric is emitted. The `SuccessPercent` metric for a step is `100` for the canary run if the step succeeds, and `0` if the step fails. The default is `true`.
- `stepDurationMetric` – Determines if a step's `Duration` metric is emitted. The `Duration` metric is emitted as a duration, in milliseconds, of the step's run. The default is `true`.

Report configurations

Includes all reports generated by CloudWatch Synthetics, such as a HAR file and a Synthetics steps report. Sensitive data redaction fields `restrictedHeaders` and `restrictedUrlParameters` also apply to logs generated by Synthetics.

- `includeRequestHeaders` – Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` – Whether to include response headers in the report. The default is `false`.
- `includeUrlPassword` – Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent the disclosure of sensitive data. The default is `false` .
- `includeRequestBody` – Whether to include the request body in the report. The default is `false`.
- `includeResponseBody` – Whether to include the response body in the report. The default is `false`.
- `restrictedHeaders` – A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing `includeRequestHeaders` as true and `restrictedHeaders` as `['Authorization']`.

- `restrictedUrlParameters` – A list of URL path or query parameters to redact. This applies to URLs that appear in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `har` – Determines if an HTTP archive (HAR) should be generated. The default is `true`.

The following is an example of a report configurations file.

```
"includeRequestHeaders": true,
"includeResponseHeaders": true,
"includeUrlPassword": false,
"includeRequestBody": true,
"includeResponseBody": true,
"restrictedHeaders": ['x-amz-security-token', 'Authorization'], // Value of these
 headers is redacted from logs and reports
"restrictedUrlParameters": ['Session', 'SigninToken'] // Values of these URL parameters
 are redacted from logs and reports
```

Logging configurations

Applies to logs generated by CloudWatch Synthetics. Controls the verbosity of request and response logs.

- `logRequest` – Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `false`.
- `logResponse` – Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `false`.
- `logRequestBody` – Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is true. The default is `false`.
- `logResponseBody` – Whether to log response bodies along with the requests in canary logs. This configuration applies only if `logResponse` is true. The default is `false`.
- `logRequestHeaders` – Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is true. The default is `false`.
- `logResponseHeaders` – Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is true. The default is `false`.

HTTP metric configurations

Configurations for metrics related to the count of network requests with different HTTP status codes, emitted by CloudWatch Synthetics for this canary.

- `metric_2xx` – Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `metric_4xx` – Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `metric_5xx` – Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` – Whether to emit the `failedRequests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` – Whether to emit the `failedRequests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated2xxMetric` – Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` – Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` – Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

Canary metric configurations

Configurations for other metrics emitted by CloudWatch Synthetics.

- `failedCanaryMetric` – Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` – Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

Other configurations

- `userAgent` – A string to append to the user agent. The user agent is a string that is included in request header, and identifies your browser to websites you visit when you use the headless browser. CloudWatch Synthetics automatically adds `CloudWatchSynthetics/`*`canary-arn`*

`to the user agent`. The specified configuration is appended to the generated user agent. The default user agent value to append is an empty string ("").

CloudWatch Synthetics environment variables

Configure the logging level and format by using environment variables.

Log format

The CloudWatch Synthetics Playwright runtime creates CloudWatch logs for every canary run. Logs are written in JSON format for convenient querying. Optionally, you can change the log format to `TEXT`.

- `Environment variable name` – CW_SYNTHETICS_LOG_FORMAT
- `Supported values` – JSON, TEXT
- `Default` – JSON

Log levels

Though enabling `Debug` mode increases verbosity, it can be useful for troubleshooting.

- `Environment variable name` – CW_SYNTHETICS_LOG_LEVEL
- `Supported values` – TRACE, DEBUG, INFO, WARN, ERROR, FATAL
- `Default` – INFO

Writing a Node.js canary script using the Puppeteer runtime

Topics

- Creating a CloudWatch Synthetics canary from scratch
- Packaging your Node.js canary files
- Changing an existing Puppeteer script to use as a Synthetics canary
- Related topics

Creating a CloudWatch Synthetics canary from scratch

Here is an example minimal Synthetics Canary script. This script passes as a successful run, and returns a string. To see what a failing canary looks like, change `let fail = false;` to `let fail = true;`.

You must define an entry point function for the canary script. To see how files are uploaded to the Amazon S3 location specified as the canary's `ArtifactS3Location`, create these files in the `/tmp` folder. All canary artifacts should be stored in `/tmp`, because it's the only writable directory. Be sure that the screenshot path is set to `/tmp` for any screenshots or other files created by the script. Synthetics automatically uploads files in `/tmp` to an S3 bucket.

```
/tmp/<name>
```

After the script runs, the pass/fail status and the duration metrics are published to CloudWatch and the files under`/tmp` are uploaded to an S3 bucket.

```
const basicCustomEntryPoint = async function () {

    // Insert your code here

    // Perform multi-step pass/fail check

    // Log decisions made and results to /tmp

    // Be sure to wait for all your code paths to complete
    // before returning control back to Synthetics.
    // In that way, your canary will not finish and report success
    // before your code has finished executing

    // Throw to fail, return to succeed
    let fail = false;
    if (fail) {
        throw "Failed basicCanary check.";
    }

    return "Successfully completed basicCanary checks.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
```

```
};
```

Next, expand the script to use Synthetics logging and make a call using the AWS SDK. For demonstration purposes, this script will create an Amazon DynamoDB client and make a call to the DynamoDB listTables API. It logs the response to the request and logs either pass or fail depending on whether the request was successful.

```
const log = require('@aws/synthetics-logger');
const AWS = require('aws-sdk');
// Require any dependencies that your script needs
// Bundle additional files and dependencies into a .zip file with folder structure
// nodejs/node_modules/additional files and folders

const basicCustomEntryPoint = async function () {

    log.info("Starting DynamoDB:listTables canary.");

    let dynamodb = new AWS.DynamoDB();
    var params = {};
    let request = await dynamodb.listTables(params);
    try {
        let response = await request.promise();
        log.info("listTables response: " + JSON.stringify(response));
    } catch (err) {
        log.error("listTables error: " + JSON.stringify(err), err.stack);
        throw err;
    }

    return "Successfully completed DynamoDB:listTables canary.";
};

exports.handler = async () => {
    return await basicCustomEntryPoint();
};
```

Packaging your Node.js canary files

For syn-nodejs-puppeteer-11.0 and above

The older packaging structure (for syn-nodejs-puppeteer-10.0 and below) is still supported in newer versions.

Create a script using one of the following options:

- .js file (CommonJS syntax)
- .mjs file (ES modules syntax)

For ES modules, use one of the following options:

- .js file (CommonJS syntax)
- .mjs file (ES modules syntax)

The package structure is defined below:

- Root-level handler file (index.js/index.mjs)
- Optional configuration file (synthetics.json)
- Additional dependencies in node_modules (if needed)

Packaging structure example:

```
  my_function/
### index.mjs
### synthetics.json
### helper-utils.mjs
### node_modules/
    ### dependencies
```

To package, follow the steps below:

1. Install dependencies (if any).

   ```
   npm install
   ```

2. Create a .zip package.

   ```
   zip -r my_deployment_package.zip
   ```

For syn-nodejs-puppeteer-11.0 and below

The following structure is required when using Amazon S3:

```
  nodejs/
### node_modules/
    ### myCanaryFilename.js
```

To add an optional sub-folder support in syn-nodejs-puppeteer-3.4+:

```
nodejs/
### node_modules/
    ### myFolder/
        ### myCanaryFilename.js
```

> **Note**
>
> Handler path in configuration must match your file location.

Handler name

Be sure to set your canary's script entry point (handler) as `myCanaryFilename.functionName` to match the file name of your script's entry point. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, then `functionName` must be `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, you can choose any function name as the handler. If you are using `syn-nodejs-puppeteer-3.4` or later, you can also optionally store the canary in a separate folder such as `nodejs/node_modules/myFolder/my_canary_filename`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Puppeteer script to use as a Synthetics canary

This section explains how to take Puppeteer scripts and modify them to run as Synthetics canary scripts. For more information about Puppeteer, see Puppeteer API v1.14.0.

Start with this example Puppeteer script:

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
```

```
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

The conversion steps are as follows:

- Create and export a `handler` function. The handler is the entry point function for the script. If you are using a runtime earlier than `syn-nodejs-puppeteer-3.4`, the handler function must be named `handler`. If you are using `syn-nodejs-puppeteer-3.4` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-nodejs-puppeteer-3.4` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

  ```
  const basicPuppeteerExample = async function () {};

  exports.handler = async () => {
      return await basicPuppeteerExample();
  };
  ```

- Use the `Synthetics` dependency.

  ```
  var synthetics = require('@aws/synthetics-puppeteer');
  ```

- Use the `Synthetics.getPage` function to get a Puppeteer `Page` object.

  ```
  const page = await synthetics.getPage();
  ```

  The page object returned by the Synthetics.getPage function has the **page.on** `request`, `response` and `requestfailed` events instrumented for logging. Synthetics also sets up HAR file generation for requests and responses on the page, and adds the canary ARN to the user-agent headers of outgoing requests on the page.

The script is now ready to be run as a Synthetics canary. Here is the updated script:

```
var synthetics = require('@aws/synthetics-puppeteer');  // Synthetics dependency

const basicPuppeteerExample = async function () {
    const page = await synthetics.getPage(); // Get instrumented page from Synthetics
    await page.goto('https://example.com');
```

```
    await page.screenshot({path: '/tmp/example.png'}); // Write screenshot to /tmp
 folder
};

exports.handler = async () => {  // Exported handler function
    return await basicPuppeteerExample();
};
```

Related topics

- Environment variables
- Encrypting environment variables at rest with a customer managed key
- Encrypting environment variables in transit
- Forcing your canary to use a static IP address

Writing a Python canary script

This script passes as a successful run, and returns a string. To see what a failing canary looks like, change fail = False to fail = True

```
def basic_custom_script():
    # Insert your code here
    # Perform multi-step pass/fail check
    # Log decisions made and results to /tmp
    # Be sure to wait for all your code paths to complete
    # before returning control back to Synthetics.
    # In that way, your canary will not finish and report success
    # before your code has finished executing
    fail = False
    if fail:
        raise Exception("Failed basicCanary check.")
    return "Successfully completed basicCanary checks."
def handler(event, context):
    return basic_custom_script()
```

Packaging your Python canary files

If you have more than one .py file or your script has a dependency, you can bundle them all into a single ZIP file. If you use the `syn-python-selenium-1.1` runtime, the ZIP file must contain your main canary .py file within a `python` folder, such as `python/my_canary_filename.py`. If

you use `syn-python-selenium-1.1` or later, you can optionally use a different folder , such as `python/myFolder/my_canary_filename.py`.

This ZIP file should contain all necessary folders and files, but the other files do not need to be in the `python` folder.

Be sure to set your canary's script entry point as `my_canary_filename.functionName` to match the file name and function name of your script's entry point. If you are using the `syn-python-selenium-1.0` runtime, then `functionName` must be `handler`. If you are using `syn-python-selenium-1.1` or later, this handler name restriction doesn't apply, and you can also optionally store the canary in a separate folder such as `python/myFolder/my_canary_filename.py`. If you store it in a separate folder, specify that path in your script entry point, such as `myFolder/my_canary_filename.functionName`.

Changing an existing Selenium script to use a Synthetics canary

You can quickly modify an existing script for Python and Selenium to be used as a canary. For more information about Selenium, see www.selenium.dev/.

For this example, start with the following Selenium script:

```
from selenium import webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

basic_selenium_script()
```

The conversion steps are as follows.

To convert a Selenium script to be used as a canary

1. Change the `import` statement to use Selenium from the `aws_synthetics` module:

   ```
   from aws_synthetics.selenium import synthetics_webdriver as webdriver
   ```

   The Selenium module from `aws_synthetics` ensures that the canary can emit metrics and logs, generate a HAR file, and work with other CloudWatch Synthetics features.

2. Create a handler function and call your Selenium method. The handler is the entry point function for the script.

   If you are using `syn-python-selenium-1.0`, the handler function must be named `handler`. If you are using `syn-python-selenium-1.1` or later, the function can have any name, but it must be the same name that is used in the script. Also, if you are using `syn-python-selenium-1.1` or later, you can store your scripts under any folder and specify that folder as part of the handler name.

   ```
   def handler(event, context):
       basic_selenium_script()
   ```

The script is now updated to be a CloudWatch Synthetics canary. Here is the updated script:

The `webdriver` is an instance of the class SyntheticsWebDriver and the browser returned by `webdriver.Chrome()` is an instance of SyntheticsBrowser.

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver

def basic_selenium_script():
    browser = webdriver.Chrome()
    browser.get('https://example.com')
    browser.save_screenshot('loaded.png')

def handler(event, context):
    basic_selenium_script()
```

Changing an existing Puppeteer Synthetics script to authenticate non-standard certificates

One important use case for Synthetics canaries is for you to monitor your own endpoints. If you want to monitor an endpoint that isn't ready for external traffic, this monitoring can sometimes mean that you don't have a proper certificate signed by a trusted third-party certificate authority.

Two possible solutions to this scenario are as follows:

- To authenticate a client certificate, see How to validate authentication using Amazon CloudWatch Synthetics - Part 2.
- To authenticate a self-signed certificate, see How to validate authentication with self-signed certificates in Amazon CloudWatch Synthetics

You are not limited to these two options when you use CloudWatch Synthetics canaries. You can extend these features and add your business logic by extending the canary code.

> **Note**
>
> Synthetics canaries running on Python runtimes innately have the `--ignore-certificate-errors` flag enabled, so those canaries shouldn't have any issues reaching sites with non-standard certificate configurations.

Writing a JSON configuration for Node.js multi Checks blueprint

The Node.js multi checks blueprint allows you to create canaries that perform multiple validation checks within a single canary run. This blueprint is useful when you want to test multiple endpoints, validate different aspects of your application, or perform a series of related checks in sequence.

Topics

- Root configuration structure
- Global settings
- Variables and data management
- Step definitions
- Check types
- Authentication methods
- Assertions and validation
- Data extraction

Root configuration structure

The root configuration defines the overall structure of your advanced API blueprint canary.

Property	Type	Required	Description
`globalSettings`	Object	No	Default configurations applied to all steps

Property	Type	Required	Description
`variables`	Object	No	Reusable values across steps (max 10)
`steps`	Object	**Yes**	Collection of monitoring steps (1-10 steps)

Example

```
{
  "globalSettings": {
    "stepTimeout": 30000,
    "userAgent": "CloudWatch-Synthetics-Advanced/1.0"
  },
  "variables": {
    "baseUrl": "https://api.example.com",
    "apiVersion": "v1"
  },
  "steps": {
    "1": {
      "stepName": "healthCheck",
      "checkerType": "HTTP",
      "url": "${baseUrl}/health",
      "httpMethod": "GET"
    }
  }
}
```

Validation rules

- Must contain at least one step
- Maximum 10 steps allowed
- No additional properties allowed beyond `globalSettings`, `variables`, and `steps`

Global settings

Global settings provide default configurations that apply to all steps unless overridden at the step level.

Properties

Property	Type	Default	Range	Description
`stepTimeout`	integer	5000	1000-30000	Default timeout for all steps (milliseconds)

Example

```
{
  "globalSettings": {
    "stepTimeout": 60000

  }
}
```

Variables and data management

Variables allow you to define reusable values that can be referenced throughout your configuration using `${variableName}` syntax.

Variable properties

Property	Type	Description
Variable names	string	Must match pattern `^[a-zA-Z][a-zA-Z0-9_]*$`
Variable values	string	Any string value

Limitations

- Maximum 10 variables per configuration
- Variable names must start with a letter
- Variable names can contain letters, numbers, and underscores only
- Maximum length not specified in schema

Example

```
{
  "variables": {
    "baseUrl": "https://api.example.com",
    "apiKey": "${AWS_SECRET:my-api-key}",
    "timeout": "30000",
    "userEmail": "test@example.com"
  }
}
```

Configuration usage

```
{
  "steps": {
    "1": {
      "url": "${baseUrl}/users",
      "timeout": "${timeout}",
      "headers": {
        "Authorization": "Bearer ${apiKey}"
      }
    }
  }
}
```

Step definitions

Steps define individual monitoring operations. Each step is numbered from 1 to 10 and contains a specific type of check.

Common step properties

Property	Type	Required	Description
`stepName`	string	**Yes**	Unique identifier for the step

Property	Type	Required	Description
checkerType	string	**Yes**	Type of check: HTTP, DNS, SSL, TCP
extractors	array	No	Data extraction configuration

Step name validation

- Pattern - ^[a-zA-Z][a-zA-Z0-9_-]*$
- Maximum length - 64 characters
- Must start with a letter

Step numbering

- Steps are numbered as string keys: "1", "2", ..., "10"
- Pattern: ^([1-9]|10)$
- Minimum 1 step required
- Maximum 10 steps allowed

Example

```
{
  "steps": {
    "1": {
      "stepName": "loginAPI",
      "checkerType": "HTTP",
      "url": "https://api.example.com/login",
      "httpMethod": "POST"
    },
    "2": {
      "stepName": "dnsCheck",
      "checkerType": "DNS",
      "domain": "example.com"
    }
  }
}
```

Check types

HTTP checks

Monitor web endpoints and APIs with comprehensive request and response validation.

Required properties

Property	Type	Description
`url`	string	Target URL (must be valid URI format)
`httpMethod`	string	HTTP method: `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, `HEAD`, `OPTIONS`

Optional properties

Property	Type	Default	Range	Description
`timeout`	integer	5000	1000-300000	Request timeout (milliseconds)
`waitTime`	integer	0	0-60	Delay before request (seconds)
`headers`	object	-	-	Custom HTTP headers
body	string	-	-	Request body for POST/PUT operations
`authentication`	object	-	-	Authentication configuration
`assertions`	array	-	-	Response validation rules

Example

```
{
  "stepName": "createUser",
  "checkerType": "HTTP",
```

```
  "url": "https://api.example.com/users",
  "httpMethod": "POST",
  "timeout": 15000,
  "headers": {
    "Content-Type": "application/json",
    "X-API-Version": "v1"
  },
  "body": "{\"name\":\"John Doe\",\"email\":\"john@example.com\"}",
  "authentication": {
    "type": "API_KEY",
    "apiKey": "${AWS_SECRET:api-credentials}",
    "headerName": "X-API-Key"
  },
  "assertions": [
    {
      "type": "STATUS_CODE",
      "operator": "EQUALS",
      "value": 201
    }
  ]
}
```

DNS checks

Validate DNS resolution and record information.

Required properties

Property	Type	Description
`domain`	string	Domain name to query (hostname format)

Optional properties

Property	Type	Default	Description
`recordType`	string	"A"	DNS record type: `A`, `CNAME`, `MX`, `TXT`, `NS`
`nameserver`	string	-	Specific DNS server to query

Property	Type	Default	Description
timeout	integer	5000	Query timeout (1000-300000ms)
port	integer	53	DNS server port (1-65535)
protocol	string	"UDP"	Protocol: UDP or TCP
assertions	array	-	DNS response validation rules

Example

```
{
  "stepName": "dnsResolution",
  "checkerType": "DNS",
  "domain": "example.com",
  "recordType": "A",
  "nameserver": "8.8.8.8",
  "timeout": 10000,
  "assertions": [
    {
      "type": "RECORD_VALUE",
      "operator": "CONTAINS",
      "value": "192.168"
    }
  ]
}
```

SSL checks

Monitor SSL certificate health and configuration.

Required properties

Property	Type	Description
hostname	string	Target hostname (hostname format)

Optional properties

Property	Type	Default	Description
`port`	integer	443	SSL port (1-65535)
`timeout`	integer	5000	Connection timeout (1000-300000ms)
`sni`	boolean	TRUE	Server Name Indication
`verifyHostname`	boolean	TRUE	Hostname verification
`allowSelfSigned`	boolean	FALSE	Accept self-signed certificates
`assertions`	array	-	Certificate validation rules

Example

```
{
  "stepName": "sslCertCheck",
  "checkerType": "SSL",
  "hostname": "secure.example.com",
  "port": 443,
  "sni": true,
  "verifyHostname": true,
  "assertions": [
    {
      "type": "CERTIFICATE_EXPIRY",
      "operator": "GREATER_THAN",
      "value": 30,
      "unit": "DAYS"
    }
  ]
}
```

TCP checks

Test TCP port connectivity and response validation.

Required properties

Property	Type	Description
hostname	string	Target hostname (hostname format)
port	integer	Target port (1-65535)

Optional properties

Property	Type	Default	Description
timeout	integer	5000	Overall timeout (1000-300000ms)
sendData	string	-	Data to send after connection
expectedResponse	string	-	Expected response data
encoding	string	"UTF-8"	Data encoding: UTF-8, ASCII, HEX
assertions	array	-	Connection and response validation

Example

```
{
  "stepName": "databaseConnection",
  "checkerType": "TCP",
  "hostname": "db.example.com",
  "port": 3306,
  "timeout": 5000,
  "sendData": "SELECT 1",
  "expectedResponse": "1",
  "assertions": [
    {
      "type": "CONNECTION_SUCCESSFUL",
      "value": true
    }
  ]
}
```

Authentication methods

No authentication

```
{
  "type": "NONE"
}
```

Basic authentication

Property	Type	Required	Description
type	string	**Yes**	Must be "BASIC"
username	string	**Yes**	Username for authentication
password	string	**Yes**	Password for authentication

Example

```
{
  "type": "BASIC",
  "username": "admin",
  "password": "${AWS_SECRET:basic-auth:password}"
}
```

API key authentication

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "API_KEY"
apiKey	string	**Yes**	-	API key value
headerName	string	No	"X-API-Key"	Header name for API key

Example

```
{
  "type": "API_KEY",
  "apiKey": "${AWS_SECRET:api-credentials}",
  "headerName": "Authorization"
}
```

OAuth client credentials

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "OAUTH_CLIENT_CREDENTIALS"
tokenUrl	string	**Yes**	-	OAuth token endpoint URL
clientId	string	**Yes**	-	OAuth client ID
clientSecret	string	**Yes**	-	OAuth client secret
scope	string	No	-	OAuth scope
audience	string	No	-	OAuth audience
resource	string	No	-	OAuth resource
tokenApiAuth	array	No	-	Token API auth methods: BASIC_AUTH_HEADER , REQUEST_BODY
tokenCacheTtl	integer	No	3600	Token cache TTL (minimum 60 seconds)

Example

```
{
  "type": "OAUTH_CLIENT_CREDENTIALS",
  "tokenUrl": "https://auth.example.com/oauth/token",
  "clientId": "${AWS_SECRET:oauth-creds:client_id}",
```

```
  "clientSecret": "${AWS_SECRET:oauth-creds:client_secret}",
  "scope": "read write",
  "tokenCacheTtl": 7200
}
```

AWS Signature (Version 4)

Property	Type	Required	Description
type	string	**Yes**	Must be "SIGV4"
service	string	**Yes**	Name of the AWS service (for example, "execute-api")
region	string	**Yes**	AWS region
roleArn	string	**Yes**	IAM role ARN for signing

Example

```
{
  "type": "SIGV4",
  "service": "execute-api",
  "region": "us-east-1",
  "roleArn": "arn:aws:iam::123456789012:role/SyntheticsRole"
}
```

Assertions and validation

HTTP assertions

Status code assertions

Property	Type	Required	Description
type	string	**Yes**	Must be "STATUS_CODE"
operator	string	**Yes**	EQUALS, NOT_EQUALS , GREATER_THAN , LESS_THAN , IN_RANGE

Property	Type	Required	Description
value	integer	Conditional	HTTP status code (100-599)
rangeMin	integer	Conditional	Minimum range value (for IN_RANGE)
rangeMax	integer	Conditional	Maximum range value (for IN_RANGE)

```
{
  "type": "STATUS_CODE",
  "operator": "EQUALS",
  "value": 200
}
```

Response time assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "RESPONSE_TIME"
operator	string	**Yes**	-	LESS_THAN , GREATER_THAN , EQUALS
value	number	**Yes**	-	Time value (minimum 0)
unit	string	No	"MILLISECONDS"	Must be "MILLISECONDS"

```
{
  "type": "RESPONSE_TIME",
  "operator": "LESS_THAN",
  "value": 500,
  "unit": "MILLISECONDS"
}
```

Head assertions

Property	Type	Required	Description
type	string	**Yes**	Must be "HEADER"
headerName	string	**Yes**	Name of header to validate
operator	string	**Yes**	EQUALS, NOT_EQUALS , CONTAINS, NOT_CONTAINS , REGEX_MATCH , EXIST
value	string/boolean	Conditional	Expected value (boolean for EXIST operator)

```
{
  "type": "HEADER",
  "headerName": "Content-Type",
  "operator": "CONTAINS",
  "value": "application/json"
}
```

Body assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "BODY"
target	string	No	"JSON"	JSON or TEXT
path	string	Conditional	-	JSONPath (required for JSON target)
operator	string	**Yes**	-	CONTAINS, NOT_CONTAINS , EQUALS, NOT_EQUALS , EXISTS
value	string/boolean	**Yes**	-	Expected value (boolean for EXISTS operator)

```
{
  "type": "BODY",
  "target": "JSON",
  "path": "$.users[0].name",
  "operator": "EQUALS",
  "value": "John Doe"
}
```

DNS assertions

Record value assertions

Property	Type	Required	Range	Description
type	string	**Yes**	-	Must be "RECORD_VALUE"
operator	string	**Yes**	-	EQUALS, NOT_EQUALS , CONTAINS, NOT_CONTAINS , REGEX_MATCH
value	string	**Yes**	-	Expected record value

Record count assertions

Property	Type	Required	Range	Description
type	string	**Yes**	-	Must be "RECORD_COUNT"
operator	string	**Yes**	-	EQUALS, GREATER_THAN , LESS_THAN
value	integer	**Yes**	≥ 0	Expected count (minimum 0)

Authoritative assertions

Property	Type	Required	Range	Description
type	string	**Yes**	-	Must be "AUTHORITATIVE"

Property	Type	Required	Range	Description
value	boolean	**Yes**	-	Expected authoritative status

TTL assertions

Property	Type	Required	Range	Description
type	string	**Yes**	-	Must be "TTL"
operator	string	**Yes**	-	EQUALS, GREATER_THAN , LESS_THAN
value	integer	**Yes**	≥ 0	Expected TTL (minimum 0)

SSL assertions

Certificate expiry assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "CERTIFICATE_EXPIRY"
operator	string	**Yes**	-	GREATER_THAN , LESS_THAN
value	integer	**Yes**	-	Time value (minimum 0)
unit	string	No	"DAYS"	DAYS, HOURS

Certificate subject assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "CERTIFICATE_SUBJECT"

Property	Type	Required	Default	Description
field	string	**Yes**	-	Subject field: CN, O, OU, C , ST, L
operator	string	**Yes**	-	CONTAINS, EQUALS, REGEX_MATCH
value	string	**Yes**	-	Expected field value

Certificate issuer assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "CERTIFICATE_ISSUER"
field	string	**Yes**	-	Issuer field: CN, O
operator	string	**Yes**	-	CONTAINS, EQUALS
value	string	**Yes**	-	Expected field value

TCP assertions

Connection success assertions

Property	Type	Required	Default	Description
type	string	**Yes**	-	Must be "CONNECTION_SUCCESSFUL"
value	boolean	**Yes**	-	Expected connection status

Response data assertions

Property	Type	Required	Default	Description
`type`	string	**Yes**	-	Must be `"RESPONSE_DATA"`
`operator`	string	**Yes**	-	`CONTAINS`, `EQUALS`, `NOT_CONTAINS` , `REGEX_MATCH` , `STARTS_WITH` , `ENDS_WITH`
`value`	string	**Yes**	-	Expected response data
`encoding`	string	No	"UTF-8"	`UTF-8`, `ASCII`, `HEX`

Data extraction

Extractors allows you to capture data from responses for use in subsequent steps or for reporting purposes.

Extraction properties

Property	Type	Required	Default	Description
`name`	string	**Yes**	-	Variable name for extracted data
`type`	string	**Yes**	-	Extraction type: `BODY`
`path`	string	No	-	JSONPath for body extraction
`regex`	string	No	-	Regular expression pattern
`regexGroup`	integer	No	0	Regex capture group (minimum 0)

Extraction name validation

- Pattern: `^[a-zA-Z][a-zA-Z0-9_]*$`
- Must start with a letter
- Can contain letters, numbers, and underscores

Limitation – Substitution does not apply for fields in the schema that have specific ENUM values

Extraction types

```
{
  "name": "userId",
  "type": "BODY",
  "path": "$.user.id"
}
```

```
{
  "stepName": "loginAndExtract",
  "checkerType": "HTTP",
  "url": "https://api.example.com/login",
  "httpMethod": "POST",
  "body": "{\"username\":\"test\",\"password\":\"pass\"}",
  "extractors": [
    {
      "name": "textVariable",
      "type": "BODY",
      "path": "$.myvalue"
    }
  ]
},
{
  "stepName": "substituteVariable",
  "checkerType": "HTTP",
  "url": "https://api.example.com/get/${textVariable}",
  "httpMethod": "GET",
  "assertions": [
    {
    "type": "BODY",
    "target": "JSON",
    "path": "$.users[0].name",
    "operator": "EQUALS",
    "value": "${textVariable}"
    }
  ]
}
```

Library functions available for canary scripts

CloudWatch Synthetics includes several built-in classes and functions that you can call when writing Node.js scripts to use as canaries.

Some apply to both UI and API canaries. Others apply to UI canaries only. A UI canary is a canary that uses the `getPage()` function and uses Puppeteer as a web driver to navigate and interact with webpages.

> **Note**
>
> Whenever you upgrade a canary to use a new version of the the Synthetics runtime, all Synthetics library functions that your canary uses are also automatically upgraded to the same version of NodeJS that the Synthetics runtime supports.

Topics

- Library functions available for Node.js canary
- Library functions available for Java canary
- Library functions available for Node.js canary scripts using Playwright
- Library functions available for Node.js canary scripts using Puppeteer
- Library functions available for Python canary scripts using Selenium

Library functions available for Node.js canary

This section describes the library functions that are available for canary scripts using the Node.js runtime.

Topics

- addExecutionError(errorMessage, ex);
- getCanaryName();
- getCanaryArn();
- getCanaryUserAgentString();
- getRuntimeVersion();
- getLogLevel();

- setLogLevel();
- executeStep(stepName, functionToExecute, [stepConfig])
- executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

addExecutionError(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `addExecutionError` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `addExecutionError` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try {await synthetics.executeStep(stepName, callbackFunc);} catch(ex)
 {synthetics.addExecutionError('Unable to take screenshot ', ex);}
```

getCanaryName();

Returns the name of the canary.

getCanaryArn();

Returns the ARN of the canary.

getCanaryUserAgentString();

Returns the custom user agent of the canary.

getRuntimeVersion();

This function is available on runtime version `syn-nodejs-3.0` and later. It returns the Synthetics runtime version of the canary. For example, the return value could be `syn-nodejs-3.0`.

getLogLevel();

Retrieves the current log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

Sets the log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
synthetics.setLogLevel(0);
```

executeStep(stepName, functionToExecute, [stepConfig])

Executes the provided step, wrapping it with start/pass/fail logging and pass/fail and duration metrics.

The `executeStep` function also does the following:

- Logs that the step started
- Starts a timer
- Executes the provided function

- When the function returns normally, it counts as passing. If the function throws, it counts as failing
- Ends the timer
- Logs whether the step passed or failed
- Emits the `stepName SuccessPercent` metric, 100 for pass or 0 for failure
- Emits the `stepName Duration metric`, with a value based on the step start and end times
- Returns what the functionToExecute returned or re-throws what `functionToExecute` threw
- Adds a step execution summary to the canary's report

Example

```
await synthetics.executeStep(stepName, async function () {
    return new Promise((resolve, reject) => {
        const req = https.request(url, (res) => {
            console.log(`Status: ${res.statusCode}`);
            if (res.statusCode >= 400) {
                reject(new Error(`Request failed with status ${res.statusCode} for
 ${url}`));
            } else {
                resolve();
            }
        });

        req.on('error', (err) => {
            reject(new Error(`Request failed for ${url}: ${err.message}`));
        });

        req.end();
    });
});
```

executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

Executes the provided HTTP request as a step, and publishes `SuccessPercent` (pass/fail) and `Duration` metrics.

executeHttpStep uses either HTTP or HTTPS native functions under the hood, depending upon the protocol specified in the request.

This function also adds a step execution summary to the canary's report. The summary includes details about each HTTP request, such as the following:

- Start time
- End time
- Status (PASSED/FAILED)
- Failure reason, if it failed
- HTTP call details such as request/response headers, body, status code, status message, and performance timings.

Topics

- Parameters
- Examples of using executeHttpStep

Parameters

stepName(*String*)

Specifies the name of the step. This name is also used for publishing CloudWatch metrics for this step.

requestOptions(*Object or String*)

The value of this parameter can be a URL, a URL string, or an object. If it is an object, then it must be a set of configurable options to make an HTTP request. It supports all options in http.request(options[, callback]) in the Node.js documentation.

In addition to these Node.js options, **requestOptions** supports the additional parameter body. You can use the body parameter to pass data as a request body.

callback(*response*)

(Optional) This is a user function which is invoked with the HTTP response. The response is of the type Class: http.IncomingMessage.

stepConfig(*object*)

(Optional) Use this parameter to override global synthetics configurations with a different configuration for this step.

Examples of using executeHttpStep

The following series of examples build on each other to illustrate the various uses of this option.

This first example configures request parameters. You can pass a URL as **requestOptions**:

```
let requestOptions = 'https://www.amazon.com';
```

Or you can pass a set of options:

```
let requestOptions = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };
```

The next example creates a callback function which accepts a response. By default, if you do not specify **callback**, CloudWatch Synthetics validates that the status is between 200 and 299 inclusive.

```
// Handle validation for positive scenario
    const callback = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };
```

The next example creates a configuration for this step that overrides the global CloudWatch Synthetics configuration. The step configuration in this example allows request headers, response headers, request body (post data), and response body in your report and restrict 'X-Amz-Security-Token' and 'Authorization' header values. By default, these values are not included in the report for security reasons. If you choose to include them, the data is only stored in your S3 bucket.

```
// By default headers, post data, and response body are not included in the report for
 security reasons.
// Change the configuration at global level or add as step configuration for individual
 steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
 values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

This final example passes your request to **executeHttpStep** and names the step.

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
 stepConfig);
```

With this set of examples, CloudWatch Synthetics adds the details from each step in your report and produces metrics for each step using **stepName**.

You will see `successPercent` and `duration` metrics for the `Verify GET products API` step. You can monitor your API performance by monitoring the metrics for your API call steps.

For a sample complete script that uses these functions, see Multi-step API canary.

Library functions available for Java canary

The `executeStep` function is used to modularize the canary code and execute it in steps. In CloudWatch Synthetics, a Synthetics step is a way to break down your canary script into a series of clearly defined actions, allowing you to monitor different parts of your application journey separately. For each step, CloudWatch Synthetics does the following:

- A report, including a summary of step execution details like the duration of a step, *pass* or *fail* status, and so on, is created for each canary run. When you choose a run in the CloudWatch Synthetics console, you can view execution details of each step on the **Step** tab.
- *SuccessPercent* and *Duration* CloudWatch metrics are emitted for each step, enabling users to monitor availability and latency of each step.

Usage

```
synthetics.executeStep(stepName,()->{
    try {
        //step code to be executed
        return null;
    } catch (Exception e) {
        throw e;
    }
}).get();
```

Parameters

- *stepName*, String (required) – A descriptive name of the Synthetics step
- *function to execute*, Callable<T> (required) – Represents the tasks to be executed
- *stepOptions*, `com.amazonaws.synthetics.StepOptions (optional)` – StepOptions object that can be used to configure the step execution.

  stepConfiguration, `com.amazonaws.synthetics.StepConfiguration`(required as part of the stepOptions)

Returns

The value returned is *CompletableFuture<T>*.

> **Note**
>
> Synthetics only supports sequential steps. Make sure to call the `.get()` method as shown in the example to make sure that the step completes before proceeding to the subsequent step.

Library functions available for Node.js canary scripts using Playwright

This section describes the library functions that are available for canary scripts using the Node.js Playwright runtime.

Topics

- launch
- newPage
- close
- getDefaultLaunchOptions
- executeStep

launch

This function launches a Chromium browser using a Playwright launch function, and returns the browser object. It decompresses browser binaries and launches the chromium browser by using default options suitable for a headless browser. For more information about the `launch` function, see `launch` in the Playwright documentation.

Usage

```
const browser = await synthetics.launch();
```

Arguments

`options` options (optional) is a configurable set of options for the browser.

Returns

Promise <`Browser`> where Browser is a Playwright browser instance.

If this function is called again, a previously-opened browser is closed before initiating a new browser. You can override launch parameters used by CloudWatch Synthetics, and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600 pixels. For more information, see Playwright launch options in the Playwright documentation.

```
const browser = await synthetics.launch({
```

```
  defaultViewport: {
      "deviceScaleFactor": 1,
      "width": 800,
      "height": 600
}});
```

You can also add or override Chromium flags passed on by default to the browser. For example, you can disable web security by adding a `--disable-web-security` flag to arguments in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
const browser = await synthetics.launch({
    args: launchArgs
  });
```

newPage

The `newPage()` function creates and returns a new Playwright page. Synthetics automatically sets up a Chrome DevTools Protocol (CDP) connection to enable network captures for HTTP archive (HAR) generation.

Usage

Use `newPage()` in either of the following ways:

1. Creating a new page in a new browser context:

```
const page = await synthetics.newPage(browser);
```

2. Creating a new page in a specified browser context:

```
// Create a new browser context
const browserContext = await browser.newContext();

// Create a new page in the specified browser context
const page = await synthetics.newPage(browserContext)
```

Arguments

Accepts either Playwright Browser instance or Playwright BrowserContext instance.

Returns

Promise <Page> where Page is a Playwright Page instance.

close

Closes the currently-opened browser.

Usage

```
await synthetics.close();
```

It is recommended to close the browser in a `finally` block of your script.

Arguments

None

Returns

Returns Promise<void> used by the Synthetics launch function for launching the browser.

getDefaultLaunchOptions

The `getDefaultLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics.

Usage

```
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

Arguments

None

Returns

Returns Playwright launch options used by the Synthetics `launch` function for launching the browser.

executeStep

The `executeStep` function is used to execute a step in a Synthetics script. In CloudWatch Synthetics, a Synthetics step is a way to break down your canary script into a series of clearly defined actions, allowing you to monitor different parts of your application journey separately. For each step, CloudWatch Synthetics does the following:

- Automatically captures a screenshot before step starts and after a step is complete. You can also capture screenshots inside a step. Screenshots are captured by default, but can be turned off by using Synthetics configurations.
- A report, including a summary, of step execution details like the duration of a step, `pass` or `fail` status, source and destination page URLs, associated screenshots, etc. is created for each canary run. When you choose a run in the CloudWatch Synthetics console, you can view execution details of each step on the **Step** tab.
- `SuccessPercent` and `Duration` CloudWatch metrics are emitted for each step, enabling users to monitor availability and latency of each step.

Usage

```
await synthetics.executeStep("mystepname", async function () {
  await page.goto(url, { waitUntil: 'load', timeout: 30000 });
}
```

> **Note**
>
> Steps should run sequentially. Be sure to use `await` on promises.

Arguments

- `stepName` string (required) (boolean)— Name of the Synthetics step.
- `functionToExecute` async function (required)— The function that you want Synthetics to run. This function should contain the logic for the step.
- `stepConfig` object (optional)— Step configuration overrides the global Synthetics configuration for this step.
  - `continueOnStepFailure` boolean (optional) — Whether to continue running the canary script after this step fails.

- `screenshotOnStepStart` boolean (optional) — Whether to take a screenshot at the start of this step.
- `screenshotOnStepSuccess` boolean (optional) — Whether to take a screenshot if this step succeeds.
- `screenshotOnStepFailure` boolean (optional) — Whether to take a screenshot if this step fails.

- `page` — Playwright page object (optional)

  A Playwright page object. Synthetics uses this page object to capture screenshots and URLs. By default, Synthetics uses the Playwright page created when the `synthetics.newPage()` function is called for capturing page details like screenshots and URLs.

Returns

Returns a Promise that resolves with the value returned by the `functionToExecute` function. For an example script, see Sample code for canary scripts in this guide.

Library functions available for Node.js canary scripts using Puppeteer

This section describes the library functions available for Node.js canary scripts.

Topics

- Node.js library classes and functions that apply to all canaries
- Node.js library classes and functions that apply to UI canaries only
- Node.js library classes and functions that apply to API canaries only

Node.js library classes and functions that apply to all canaries

The following CloudWatch Synthetics library functions for Node.js are useful for all canaries.

Topics

- Synthetics class
- SyntheticsConfiguration class
- Synthetics logger
- SyntheticsLogHelper class

Synthetics class

The following functions for all canaries are in the Synthetics class.

Topics

- addExecutionError(errorMessage, ex);
- getCanaryName();
- getCanaryArn();
- getCanaryUserAgentString();
- getRuntimeVersion();
- getLogLevel();
- setLogLevel();

addExecutionError(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `addExecutionError` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `addExecutionError` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try {
    await synthetics.takeScreenshot(stepName, "loaded");
} catch(ex) {
    synthetics.addExecutionError('Unable to take screenshot ', ex);
}
```

getCanaryName();

Returns the name of the canary.

getCanaryArn();

Returns the ARN of the canary.

getCanaryUserAgentString();

Returns the custom user agent of the canary.

getRuntimeVersion();

This function is available in runtime version `syn-nodejs-puppeteer-3.0` and later. It returns the Synthetics runtime version of the canary. For example, the return value could be `syn-nodejs-puppeteer-3.0`.

getLogLevel();

Retrieves the current log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
let logLevel = synthetics.getLogLevel();
```

setLogLevel();

Sets the log level for the Synthetics library. Possible values are the following:

- `0` – Debug
- 1 – Info
- 2 – Warn
- 3 – Error

Example:

```
synthetics.setLogLevel(0);
```

SyntheticsConfiguration class

This class is available only in the `syn-nodejs-2.1` runtime version or later.

You can use the `SyntheticsConfiguration` class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level, which are applied to all steps of canaries. You can also override these configurations at the step level by passing configuration key and value pairs.

You can pass in options at the step level. For examples, see async executeStep(stepName, functionToExecute, [stepConfig]); and executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

Topics

- setConfig(options)
- Visual monitoring

setConfig(options)

`options` is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *`options`* .

setConfig(options) for all canaries

For canaries using `syn-nodejs-puppeteer-3.2` or later, the **(options)** for **setConfig** can include the following parameters:

- `includeRequestHeaders` (boolean)— Whether to include request headers in the report. The default is `false`.
- `includeResponseHeaders` (boolean)— Whether to include response headers in the report. The default is `false`.
- `restrictedHeaders` (array)— A list of header values to ignore, if headers are included. This applies to both request and response headers. For example, you can hide your credentials by passing **includeRequestHeaders** as `true` and **restrictedHeaders** as `['Authorization']`.

- `includeRequestBody` (boolean)— Whether to include the request body in the report. The default is `false`.
- `includeResponseBody` (boolean)— Whether to include the response body in the report. The default is `false`.

> **Important**
>
> If you enable either `includeResponseBody` or `logResponseBody`, the data object is not returned in the response from some APIs, such as aws-sdk v3 clients. This is because of a limitation of Node.js and the type of response object used.

setConfig(options) regarding CloudWatch metrics

For canaries using `syn-nodejs-puppeteer-3.1` or later, the **(options)** for **setConfig** can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see CloudWatch metrics published by canaries.

- `failedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failedRequestsMetric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_2xxMetric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_4xxMetric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `_5xxMetric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `stepDurationMetric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.

- `stepSuccessMetric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName` `StepName` dimensions) for this canary. The default is `true`.
- `aggregatedFailedCanaryMetric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregatedFailedRequestsMetric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated2xxMetric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated4xxMetric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated5xxMetric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `visualMonitoringSuccessPercentMetric` (boolean)— Whether to emit the `visualMonitoringSuccessPercent` metric for this canary. The default is `true`.
- `visualMonitoringTotalComparisonsMetric` (boolean)— Whether to emit the `visualMonitoringTotalComparisons` metric for this canary. The default is `false`.
- `includeUrlPassword` (boolean)— Whether to include a password that appears in the URL. By default, passwords that appear in URLs are redacted from logs and reports, to prevent disclosing sensitive data. The default is `false`.
- `restrictedUrlParameters` (array)— A list of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. The parameter is case-insensitive. You can pass an asterisk (*) as a value to redact all URL path and query parameter values. The default is an empty array.
- `logRequest` (boolean)— Whether to log every request in canary logs. For UI canaries, this logs each request sent by the browser. The default is `true`.
- `logResponse` (boolean)— Whether to log every response in canary logs. For UI canaries, this logs every response received by the browser. The default is `true`.
- `logRequestBody` (boolean)— Whether to log request bodies along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false` .
- `logResponseBody` (boolean)— Whether to log response bodies along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

> **Important**
>
> If you enable either `includeResponseBody` or `logResponseBody`, the data object is not returned in the response from some APIs, such as aws-sdk v3 clients. This is because of a limitation of Node.js and the type of response object used.

- `logRequestHeaders` (boolean)— Whether to log request headers along with the requests in canary logs. This configuration applies only if `logRequest` is `true`. The default is `false`.

  Note that `includeRequestHeaders` enables headers in artifacts.
- `logResponseHeaders` (boolean)— Whether to log response headers along with the responses in canary logs. This configuration applies only if `logResponse` is `true`. The default is `false`.

  Note that `includeResponseHeaders` enables headers in artifacts.

> **Note**
>
> The `Duration` and `SuccessPercent` metrics are always emitted for each canary, both with and without the `CanaryName` metric.

Methods to enable or disable metrics

disableAggregatedRequestMetrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disableRequestMetrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disableStepMetrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enableAggregatedRequestMetrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enableRequestMetrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enableStepMetrics()

Enables all step metrics, including both step success metrics and step duration metrics.

get2xxMetric()

Returns whether the canary emits a `2xx` metric with the `CanaryName` dimension.

get4xxMetric()

Returns whether the canary emits a `4xx` metric with the `CanaryName` dimension.

get5xxMetric()

Returns whether the canary emits a `5xx` metric with the `CanaryName` dimension.

getAggregated2xxMetric()

Returns whether the canary emits a `2xx` metric with no dimension.

getAggregated4xxMetric()

Returns whether the canary emits a `4xx` metric with no dimension.

getAggregatedFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with no dimension.

getAggregatedFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with no dimension.

getAggregated5xxMetric()

Returns whether the canary emits a `5xx` metric with no dimension.

getFailedCanaryMetric()

Returns whether the canary emits a `Failed` metric with the `CanaryName` dimension.

getFailedRequestsMetric()

Returns whether the canary emits a `Failed requests` metric with the `CanaryName` dimension.

getStepDurationMetric()

Returns whether the canary emits a `Duration` metric with the `CanaryName` dimension for this canary.

getStepSuccessMetric()

Returns whether the canary emits a `StepSuccess` metric with the `CanaryName` dimension for this canary.

with2xxMetric(_2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with4xxMetric(_4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with5xxMetric(_5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

withAggregated5xxMetric(aggregated5xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

withAggregatedFailedCanaryMetric(aggregatedFailedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

withAggregatedFailedRequestsMetric(aggregatedFailedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

withFailedCanaryMetric(failedCanaryMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

withFailedRequestsMetric(failedRequestsMetric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

withStepDurationMetric(stepDurationMetric)

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

withStepSuccessMetric(stepSuccessMetric)

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable other features

withHarFile()

Accepts a Boolean argument, which specifies whether to create a HAR file for this canary.

withStepsReport()

Accepts a Boolean argument, which specifies whether to report a step execution summary for this canary.

withIncludeUrlPassword()

Accepts a Boolean argument, which specifies whether to include passwords that appear in URLs in logs and reports.

withRestrictedUrlParameters()

Accepts an array of URL path or query parameters to redact. This applies to URLs appearing in logs, reports, and errors. You can pass an asterisk (*) as a value to redact all URL path and query parameter values

withLogRequest()

Accepts a Boolean argument, which specifies whether to log every request in the canary's logs.

withLogResponse()

Accepts a Boolean argument, which specifies whether to log every response in the canary's logs.

withLogRequestBody()

Accepts a Boolean argument, which specifies whether to log every request body in the canary's logs.

withLogResponseBody()

Accepts a Boolean argument, which specifies whether to log every response body in the canary's logs.

withLogRequestHeaders()

Accepts a Boolean argument, which specifies whether to log every request header in the canary's logs.

withLogResponseHeaders()

Accepts a Boolean argument, which specifies whether to log every response header in the canary's logs.

getHarFile()

Returns whether the canary creates a HAR file.

getStepsReport()

Returns whether the canary reports a step execution summary.

getIncludeUrlPassword()

Returns whether the canary includes passwords that appear in URLs in logs and reports.

getRestrictedUrlParameters()

Returns whether the canary redacts URL path or query parameters.

getLogRequest()

Returns whether the canary logs every request in the canary's logs.

getLogResponse()

Returns whether the canary logs every response in the canary's logs.

getLogRequestBody()

Returns whether the canary logs every request body in the canary's logs.

getLogResponseBody()

Returns whether the canary logs every response body in the canary's logs.

getLogRequestHeaders()

Returns whether the canary logs every request header in the canary's logs.

getLogResponseHeaders()

Returns whether the canary logs every response header in the canary's logs.

Functions for all canaries

- `withIncludeRequestHeaders`(includeRequestHeaders)
- `withIncludeResponseHeaders`(includeResponseHeaders)
- `withRestrictedHeaders`(restrictedHeaders)
- `withIncludeRequestBody`(includeRequestBody)
- `withIncludeResponseBody`(includeResponseBody)
- `enableReportingOptions`()— Enables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .

- `disableReportingOptions()`— Disables all reporting options-- **includeRequestHeaders**, **includeResponseHeaders**, **includeRequestBody**, and **includeResponseBody**, .

setConfig(options) for UI canaries

For UI canaries, **setConfig** can include the following Boolean parameters:

- `continueOnStepFailure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the **executeStep** function). If any steps fail, the canary run will still be marked as failed. The default is `false`.
- `harFile` (boolean)— Whether to create a HAR file. The default is `True`.
- `screenshotOnStepStart` (boolean)— Whether to take a screenshot before starting a step.
- `screenshotOnStepSuccess` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshotOnStepFailure` (boolean)— Whether to take a screenshot after a step fails.

Methods to enable or disable screenshots

disableStepScreenshots()

Disables all screenshot options (screenshotOnStepStart, screenshotOnStepSuccess, and screenshotOnStepFailure).

enableStepScreenshots()

Enables all screenshot options (screenshotOnStepStart, screenshotOnStepSuccess, and screenshotOnStepFailure). By default, all these methods are enabled.

getScreenshotOnStepFailure()

Returns whether the canary takes a screenshot after a step fails.

getScreenshotOnStepStart()

Returns whether the canary takes a screenshot before starting a step.

getScreenshotOnStepSuccess()

Returns whether the canary takes a screenshot after completing a step successfully.

withScreenshotOnStepStart(screenshotOnStepStart)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

withScreenshotOnStepSuccess(screenshotOnStepSuccess)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

withScreenshotOnStepFailure(screenshotOnStepFailure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration.

```
// Import Synthetics dependency
const synthetics = require('@aws/synthetics-puppeteer');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();
```

Then, set the configuration for each option by calling the setConfig method using one of the following options.

```
// Set configuration values
    synConfig.setConfig({
        screenshotOnStepStart: true,
        screenshotOnStepSuccess: false,
        screenshotOnStepFailure: false
    });
```

Or

```
synConfig.withScreenshotOnStepStart(false).withScreenshotOnStepSuccess(true).withScreenshotOnSt
```

To disable all screenshots, use the `disableStepScreenshots()` function as in this example.

```
synConfig.disableStepScreenshots();
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

setConfig(options) for API canaries

For API canaries, **setConfig** can include the following Boolean parameters:

- `continueOnHttpStepFailure` (boolean)— Whether to continue with running the canary script after an HTTP step fails (this refers to the **executeHttpStep** function). If any steps fail, the canary run will still be marked as failed. The default is `true`.

Visual monitoring

Visual monitoring compares screenshots taken during a canary run with screenshots taken during a baseline canary run. If the discrepancy between the two screenshots is beyond a threshold percentage, the canary fails and you can see the areas with differences highlighted in color in the canary run report. Visual monitoring is supported in canaries running **syn-puppeteer-node-3.2** and later. It is not currently supported in canaries running Python and Selenium.

To enable visual monitoring, add the following line of code to the canary script. For more details, see SyntheticsConfiguration class.

```
syntheticsConfiguration.withVisualCompareWithBaseRun(true);
```

The first time that the canary runs successfully after this line is added to the script, it uses the screenshots taken during that run as the baseline for comparison. After that first canary run, you can use the CloudWatch console to edit the canary to do any of the following:

- Set the next run of the canary as the new baseline.
- Draw boundaries on the current baseline screenshot to designate areas of the screenshot to ignore during visual comparisons.
- Remove a screenshot from being used for visual monitoring.

For more information about using the CloudWatch console to edit a canary, see Edit or delete a canary.

Other options for visual monitoring

syntheticsConfiguration.withVisualVarianceThresholdPercentage(desiredPercentage)

Set the acceptable percentage for screenshot variance in visual comparisons.

syntheticsConfiguration.withVisualVarianceHighlightHexColor("#fafa00")

Set the highlight color that designates variance areas when you look at canary run reports that use visual monitoring.

syntheticsConfiguration.withFailCanaryRunOnVisualVariance(failCanary)

Set whether or not the canary fails when there is a visual difference that is more than the threshold. The default is to fail the canary.

Synthetics logger

SyntheticsLogger writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

You can use the SyntheticsLogger, assuming you want to run the Synthetics Library at the same log level as your Synthetics canary logging.

Using the SyntheticsLogger is not required to create a log file that is uploaded to your S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in S3 as artifacts.

To use the Synthetics Library logger:

```
const log = require('@aws/synthetics-logger');
```

Useful function definitions:

log.debug(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.debug("Starting step - login.");
```

log.error(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
try {
  await login();
catch (ex) {
  log.error("Error encountered in step - login.", ex);
}
```

log.info(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.info("Successfully completed step - login.");
```

log.log(*message*, *ex*);

This is an alias for `log.info`.

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
 log.log("Successfully completed step - login.");
```

log.warn(*message*, *ex*);

Parameters: *message* is the message to log. *ex* is the exception, if any, to log.

Example:

```
log.warn("Exception encountered trying to publish CloudWatch Metric.", ex);
```

SyntheticsLogHelper class

The `SyntheticsLogHelper` class is available in the runtime `syn-nodejs-puppeteer-3.2` and later runtimes. It is already initialized in the CloudWatch Synthetics library and is configured with Synthetics configuration. You can add this as a dependency in your script. This class enables you to sanitize URLs, headers, and error messages to redact sensitive information.

> **Note**
>
> Synthetics sanitizes all URLs and error messages it logs before including them in logs, reports, HAR files, and canary run errors based on the Synthetics configuration setting `restrictedUrlParameters`. You have to use `getSanitizedUrl` or `getSanitizedErrorMessage` only if you are logging URLs or errors in your script. Synthetics does not store any canary artifacts except for canary errors thrown by the script. Canary run artifacts are stored in your customer account. For more information, see Security considerations for Synthetics canaries.

Topics

- getSanitizedUrl(url, stepConfig = null)
- getSanitizedErrorMessage
- getSanitizedHeaders(headers, stepConfig=null)

getSanitizedUrl(url, stepConfig = null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized url strings based on the configuration. You can choose to redact sensitive URL parameters such as password and access_token by setting the property `restrictedUrlParameters`. By default, passwords in URLs are redacted. You can enable URL passwords if needed by setting `includeUrlPassword` to true.

This function throws an error if the URL passed in is not a valid URL.

Parameters

- *`url`* is a string and is the URL to sanitize.
- *`stepConfig`* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following sample URL: `https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`. In this example, `access_token` contains your sensitive information which shouldn't be logged. Note that the Synthetics services doesn't store any canary run artifacts. Artifacts such as logs, screenshots, and reports are all stored in an Amazon S3 bucket in your customer account.

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('@aws/synthetics-puppeteer');

// Import Synthetics logger for logging url
const log = require('@aws/synthetics-logger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('@aws/synthetics-log-helper');

const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('URL');



const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('URL', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

Next, sanitize and log the URL

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('@aws/synthetics-log-helper');
```

```
const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('https://example.com/learn/
home?access_token=12345&token_type=Bearer&expires_in=1200');
```

This logs the following in your canary log.

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

You can override the Synthetics configuration for a URL by passing in an optional parameter containing Synthetics configuration options, as in the following example.

```
const urlConfig = {
   restrictedUrlParameters = ['*']
};
const sanitizedUrl = syntheticsLogHelper.getSanitizedUrl('https://example.com/learn/
home?access_token=12345&token_type=Bearer&expires_in=1200', urlConfig);
logger.info('My example url is: ' + sanitizedUrl);
```

The preceding example redacts all query parameters, and is logged as follows:

```
My example url is: https://example.com/learn/home?
access_token=REDACTED&token_type=REDACTED&expires_in=REDACTED
```

getSanitizedErrorMessage

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized error strings by sanitizing any URLs present based on the Synthetics configuration. You can choose to override the global Synthetics configuration when you call this function by passing an optional `stepConfig` parameter.

Parameters

- *`error`* is the error to sanitize. It can be an Error object or a string.
- *`stepConfig`* (Optional) overrides the global Synthetics configuration for this function. If `stepConfig` is not passed in, the global configuration is used to sanitize the URL.

Example

This example uses the following error: `Failed to load url: https://example.com/learn/home?access_token=12345&token_type=Bearer&expires_in=1200`

The first step is to set the Synthetics configuration.

```
// Import Synthetics dependency
const synthetics = require('@aws/synthetics-puppeteer');

// Import Synthetics logger for logging url
const log = require('@aws/synthetics-logger');

// Get Synthetics configuration
const synConfig = synthetics.getConfiguration();

// Set restricted parameters
synConfig.setConfig({
   restrictedUrlParameters: ['access_token'];
});
```

Next, sanitize and log the error message

```
// Import SyntheticsLogHelper dependency
const syntheticsLogHelper = require('@aws/synthetics-log-helper');

try {
   // Your code which can throw an error containing url which your script logs
} catch (error) {
    const sanitizedErrorMessage =
 syntheticsLogHelper.getSanitizedErrorMessage(errorMessage);
    logger.info(sanitizedErrorMessage);
}
```

This logs the following in your canary log.

```
Failed to load url: https://example.com/learn/home?
access_token=REDACTED&token_type=Bearer&expires_in=1200
```

getSanitizedHeaders(headers, stepConfig=null)

This function is available in `syn-nodejs-puppeteer-3.2` and later. It returns sanitized headers based on the `restrictedHeaders` property of `syntheticsConfiguration`. The headers specified in the `restrictedHeaders` property are redacted from logs, HAR files, and reports.

Parameters

- *headers* is an object containing the headers to sanitize.
- *stepConfig* (Optional) overrides the global Synthetics configuration for this function. If stepConfig is not passed in, the global configuration is used to sanitize the headers.

Node.js library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for UI canaries.

Topics

- Synthetics class
- BrokenLinkCheckerReport class
- SyntheticsLink class

Synthetics class

The following functions are in the Synthetics class.

Topics

- async addUserAgent(page, userAgentString);
- async executeStep(stepName, functionToExecute, [stepConfig]);
- getDefaultLaunchOptions();
- getPage();
- getRequestResponseLogHelper();
- launch(options)
- RequestResponseLogHelper class
- setRequestResponseLogHelper();
- async takeScreenshot(name, suffix);

async addUserAgent(page, userAgentString);

This function appends *userAgentString* to the specified page's user-agent header.

Example:

```
await synthetics.addUserAgent(page, "MyApp-1.0");
```

Results in the page's user-agent header being set to *browsers-user-agent-header-value*MyApp-1.0

async executeStep(stepName, functionToExecute, [stepConfig]);

Executes the provided step, wrapping it with start/pass/fail logging, start/pass/fail screenshots, and pass/fail and duration metrics.

> **Note**
>
> If you are using the syn-nodejs-2.1 or later runtime, you can configure whether and when screenshots are taken. For more information, see SyntheticsConfiguration class.

The executeStep function also does the following:

- Logs that the step started.
- Takes a screenshot named <stepName>-starting.
- Starts a timer.
- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named <stepName>-succeeded or <stepName>-failed.
- Emits the stepName SuccessPercent metric, 100 for pass or 0 for failure.
- Emits the stepName Duration metric, with a value based on the step start and end times.
- Finally, returns what the functionToExecute returned or re-throws what functionToExecute threw.

If the canary uses the syn-nodejs-2.0 runtime or later, this function also adds a step execution summary to the canary's report. The summary includes details about each step, such as start time, end time, status (PASSED/FAILED), failure reason (if failed), and screenshots captured during the execution of each step.

Example:

```
await synthetics.executeStep('navigateToUrl', async function (timeoutInMillis = 30000)
 {
           await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});});
```

Response:

Returns what `functionToExecute` returns.

Updates with syn-nodejs-2.2

Starting with `syn-nodejs-2.2`, you can optionally pass step configurations to override CloudWatch Synthetics configurations at the step level. For a list of options that you can pass to `executeStep`, see SyntheticsConfiguration class.

The following example overrides the default `false` configuration for `continueOnStepFailure` to `true` and specifies when to take screenthots.

```
var stepConfig = {
    'continueOnStepFailure': true,
    'screenshotOnStepStart': false,
    'screenshotOnStepSuccess': true,
    'screenshotOnStepFailure': false
}

await executeStep('Navigate to amazon', async function (timeoutInMillis = 30000) {
      await page.goto(url, {waitUntil: ['load', 'networkidle0'], timeout:
 timeoutInMillis});
 }, stepConfig);
```

getDefaultLaunchOptions();

The `getDefaultLaunchOptions()` function returns the browser launch options that are used by CloudWatch Synthetics. For more information, see Launch options type

```
// This function returns default launch options used by Synthetics.
const defaultOptions = await synthetics.getDefaultLaunchOptions();
```

getPage();

Returns the current open page as a Puppeteer object. For more information, see Puppeteer API v1.14.0.

Example:

```
let page = await synthetics.getPage();
```

Response:

The page (Puppeteer object) that is currently open in the current browser session.

getRequestResponseLogHelper();

> **Important**
>
> In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use RequestResponseLogHelper class instead.

Use this function as a builder pattern for tweaking the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper().withLogRequestHeaders(fals
```

Response:

```
{RequestResponseLogHelper}
```

launch(options)

The options for this function are available only in the `syn-nodejs-2.1` runtime version or later.

This function is used only for UI canaries. It closes the existing browser and launches a new one.

> **Note**
>
> CloudWatch Synthetics always launches a browser before starting to run your script. You don't need to call launch() unless you want to launch a new browser with custom options.

(options) is a configurable set of options to set on the browser. For more information, Launch options type .

If you call this function with no options, Synthetics launches a browser with default arguments, `executablePath`, and `defaultViewport`. The default viewport in CloudWatch Synthetics is 1920 by 1080.

You can override launch parameters used by CloudWatch Synthetics and pass additional parameters when launching the browser. For example, the following code snippet launches a browser with default arguments and a default executable path, but with a viewport of 800 x 600.

```
await synthetics.launch({
        defaultViewport: {
            "deviceScaleFactor": 1,
            "width": 800,
            "height": 600
    }});
```

The following sample code adds a new `ignoreHTTPSErrors` parameter to the CloudWatch Synthetics launch parameters:

```
await synthetics.launch({
        ignoreHTTPSErrors: true
 });
```

You can disable web security by adding a `--disable-web-security` flag to args in the CloudWatch Synthetics launch parameters:

```
// This function adds the --disable-web-security flag to the launch parameters
const defaultOptions = await synthetics.getDefaultLaunchOptions();
const launchArgs = [...defaultOptions.args, '--disable-web-security'];
await synthetics.launch({
     args: launchArgs
  });
```

RequestResponseLogHelper class

> **Important**
>
> In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this class is deprecated. Any use of this class causes a warning to appear in your canary logs. This

function will be removed in future runtime versions. If you are using this function, use RequestResponseLogHelper class instead.

Handles the fine-grained configuration and creation of string representations of request and response payloads.

```
class RequestResponseLogHelper {

    constructor () {
        this.request = {url: true, resourceType: false, method: false, headers: false,
 postData: false};
        this.response = {status: true, statusText: true, url: true, remoteAddress:
 false, headers: false};
    }

    withLogRequestUrl(logRequestUrl);

    withLogRequestResourceType(logRequestResourceType);

    withLogRequestMethod(logRequestMethod);

    withLogRequestHeaders(logRequestHeaders);

    withLogRequestPostData(logRequestPostData);


    withLogResponseStatus(logResponseStatus);

    withLogResponseStatusText(logResponseStatusText);

    withLogResponseUrl(logResponseUrl);

    withLogResponseRemoteAddress(logResponseRemoteAddress);

    withLogResponseHeaders(logResponseHeaders);
```

Example:

```
synthetics.setRequestResponseLogHelper(getRequestResponseLogHelper()
.withLogRequestPostData(true)
.withLogRequestHeaders(true)
```

```
.withLogResponseHeaders(true));
```

Response:

```
{RequestResponseLogHelper}
```

setRequestResponseLogHelper();

> **Important**
>
> In canaries that use the `syn-nodejs-puppeteer-3.2` runtime or later, this function is deprecated along with the `RequestResponseLogHelper` class. Any use of this function causes a warning to appear in your canary logs. This function will be removed in future runtime versions. If you are using this function, use RequestResponseLogHelper class instead.

Use this function as a builder pattern for setting the request and response logging flags.

Example:

```
synthetics.setRequestResponseLogHelper().withLogRequestHeaders(true).withLogResponseHeaders(tru
```

Response:

```
{RequestResponseLogHelper}
```

async takeScreenshot(name, suffix);

Takes a screenshot (.PNG) of the current page with name and suffix (optional).

Example:

```
await synthetics.takeScreenshot("navigateToUrl", "loaded")
```

This example captures and uploads a screenshot named `01-navigateToUrl-loaded.png` to the canary's S3 bucket.

You can take a screenshot for a particular canary step by passing the `stepName` as the first parameter. Screenshots are linked to the canary step in your reports, to help you track each step while debugging.

CloudWatch Synthetics canaries automatically take screenshots before starting a step (the `executeStep` function) and after the step completion (unless you configure the canary to disable screenshots). You can take more screenshots by passing in the step name in the `takeScreenshot` function.

The following example takes screenshot with the `signupForm` as the value of the `stepName`. The screenshot will be named `02-signupForm-address` and will be linked to the step named `signupForm` in the canary report.

```
await synthetics.takeScreenshot('signupForm', 'address')
```

BrokenLinkCheckerReport class

This class provides methods to add a synthetics link. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `BrokenLinkCheckerReport`, include the following lines in the script:

```
const BrokenLinkCheckerReport = require('@aws/synthetics-broken-link-checker-report');

const brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

Useful function definitions:

addLink(*syntheticsLink*, isBroken)

`syntheticsLink` is a `SyntheticsLink` object representing a link. This function adds the link according to the status code. By default, it considers a link to be broken if the status code is not available or the status code is 400 or higher. You can override this default behavior by passing in the optional parameter `isBrokenLink` with a value of `true` or `false`.

This function does not have a return value.

getLinks()

This function returns an array of `SyntheticsLink` objects that are included in the broken link checker report.

getTotalBrokenLinks()

This function returns a number representing the total number of broken links.

getTotalLinksChecked()

This function returns a number representing the total number of links included in the report.

How to use BrokenLinkCheckerReport

The following canary script code snippet demonstrates an example of navigating to a link and adding it to the broken link checker report.

1. Import `SyntheticsLink`, `BrokenLinkCheckerReport`, and `Synthetics`.

```
const BrokenLinkCheckerReport = require('@aws/synthetics-broken-link-checker-
report');
const SyntheticsLink = require('@aws/synthetics-link');

// Synthetics dependency
const synthetics = require('@aws/synthetics-puppeteer');
```

2. To add a link to the report, create an instance of `BrokenLinkCheckerReport`.

```
let brokenLinkCheckerReport = new BrokenLinkCheckerReport();
```

3. Navigate to the URL and add it to the broken link checker report.

```
let url = "https://amazon.com";

let syntheticsLink = new SyntheticsLink(url);

// Navigate to the url.
let page = await synthetics.getPage();

// Create a new instance of Synthetics Link
let link = new SyntheticsLink(url)

try {
    const response = await page.goto(url, {waitUntil: 'domcontentloaded', timeout:
 30000});
} catch (ex) {
    // Add failure reason if navigation fails.
```

```
    link.withFailureReason(ex);
}

if (response) {
    // Capture screenshot of destination page
    let screenshotResult = await synthetics.takeScreenshot('amazon-home', 'loaded');

    // Add screenshot result to synthetics link
    link.addScreenshotResult(screenshotResult);

    // Add status code and status description to the link
    link.withStatusCode(response.status()).withStatusText(response.statusText())
}

// Add link to broken link checker report.
brokenLinkCheckerReport.addLink(link);
```

4. Add the report to Synthetics. This creates a JSON file named `BrokenLinkCheckerReport.json` in your S3 bucket for each canary run. You can see a links report in the console for each canary run along with screenshots, logs, and HAR files.

```
await synthetics.addReport(brokenLinkCheckerReport);
```

SyntheticsLink class

This class provides methods to wrap information. It's supported only on canaries that use the `syn-nodejs-2.0-beta` version of the runtime or later.

To use `SyntheticsLink`, include the following lines in the script:

```
const SyntheticsLink = require('@aws/synthetics-link');

const syntheticsLink = new SyntheticsLink("https://www.amazon.com");
```

This function returns `syntheticsLink`*`Object`*

Useful function definitions:

withUrl(*`url`*)

`url` is a URL string. This function returns `syntheticsLink`*`Object`*

withText(*text*)

text is a string representing anchor text. This function returns `syntheticsLink`*Object*. It adds anchor text corresponding to the link.

withParentUrl(*parentUrl*)

parentUrl is a string representing the parent (source page) URL. This function returns `syntheticsLink` *Object*

withStatusCode(*statusCode*)

statusCode is a string representing the status code. This function returns `syntheticsLink`*Object*

withFailureReason(*failureReason*)

failureReason is a string representing the failure reason. This function returns `syntheticsLink` *Object*

addScreenshotResult(*screenshotResult*)

screenshotResult is an object. It is an instance of `ScreenshotResult` that was returned by the Synthetics function `takeScreenshot`. The object includes the following:

- `fileName`— A string representing the `screenshotFileName`
- `pageUrl` (optional)
- `error` (optional)

Node.js library classes and functions that apply to API canaries only

The following CloudWatch Synthetics library functions for Node.js are useful only for API canaries.

Topics

- executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

executeHttpStep(stepName, requestOptions, [callback], [stepConfig])

Executes the provided HTTP request as a step, and publishes `SuccessPercent` (pass/fail) and `Duration` metrics.

executeHttpStep uses either HTTP or HTTPS native functions under the hood, depending upon the protocol specified in the request.

This function also adds a step execution summary to the canary's report. The summary includes details about each HTTP request, such as the following:

- Start time
- End time
- Status (PASSED/FAILED)
- Failure reason, if it failed
- HTTP call details such as request/response headers, body, status code, status message, and performance timings.

Topics

- Parameters
- Examples of using executeHttpStep

Parameters

stepName(*String*)

Specifies the name of the step. This name is also used for publishing CloudWatch metrics for this step.

requestOptions(*Object or String*)

The value of this parameter can be a URL, a URL string, or an object. If it is an object, then it must be a set of configurable options to make an HTTP request. It supports all options in http.request(options[, callback]) in the Node.js documentation.

In addition to these Node.js options, **requestOptions** supports the additional parameter body. You can use the body parameter to pass data as a request body.

callback(*response*)

(Optional) This is a user function which is invoked with the HTTP response. The response is of the type Class: http.IncomingMessage.

stepConfig(*object*)

(Optional) Use this parameter to override global synthetics configurations with a different configuration for this step.

Examples of using executeHttpStep

The following series of examples build on each other to illustrate the various uses of this option.

This first example configures request parameters. You can pass a URL as **requestOptions**:

```
let requestOptions = 'https://www.amazon.com';
```

Or you can pass a set of options:

```
let requestOptions = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };
```

The next example creates a callback function which accepts a response. By default, if you do not specify **callback**, CloudWatch Synthetics validates that the status is between 200 and 299 inclusive.

```
// Handle validation for positive scenario
    const callback = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
```

```
    };
```

The next example creates a configuration for this step that overrides the global CloudWatch Synthetics configuration. The step configuration in this example allows request headers, response headers, request body (post data), and response body in your report and restrict 'X-Amz-Security-Token' and 'Authorization' header values. By default, these values are not included in the report for security reasons. If you choose to include them, the data is only stored in your S3 bucket.

```
// By default headers, post data, and response body are not included in the report for
 security reasons.
// Change the configuration at global level or add as step configuration for individual
 steps
let stepConfig = {
    includeRequestHeaders: true,
    includeResponseHeaders: true,
    restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted header
 values do not appear in report generated.
    includeRequestBody: true,
    includeResponseBody: true
};
```

This final example passes your request to **executeHttpStep** and names the step.

```
await synthetics.executeHttpStep('Verify GET products API', requestOptions, callback,
 stepConfig);
```

With this set of examples, CloudWatch Synthetics adds the details from each step in your report and produces metrics for each step using **stepName**.

You will see `successPercent` and `duration` metrics for the `Verify GET products API` step. You can monitor your API performance by monitoring the metrics for your API call steps.

For a sample complete script that uses these functions, see Multi-step API canary.

Library functions available for Python canary scripts using Selenium

This section lists the Selenium library functions available for Python canary scripts.

Topics

- Python and Selenium library classes and functions that apply to all canaries
- Python and Selenium library classes and functions that apply to UI canaries only

Python and Selenium library classes and functions that apply to all canaries

The following CloudWatch Synthetics Selenium library functions for Python are useful for all canaries.

Topics

- SyntheticsConfiguration class
- SyntheticsLogger class

SyntheticsConfiguration class

You can use the SyntheticsConfiguration class to configure the behavior of Synthetics library functions. For example, you can use this class to configure the `executeStep()` function to not capture screenshots.

You can set CloudWatch Synthetics configurations at the global level.

Function definitions:

set_config(options)

```
from aws_synthetics.common import synthetics_configuration
```

`options` is an object, which is a set of configurable options for your canary. The following sections explain the possible fields in *`options`* .

- `screenshot_on_step_start` (boolean)— Whether to take a screenshot before starting a step.
- `screenshot_on_step_success` (boolean)— Whether to take a screenshot after completing a successful step.
- `screenshot_on_step_failure` (boolean)— Whether to take a screenshot after a step fails.

with_screenshot_on_step_start(screenshot_on_step_start)

Accepts a Boolean argument, which indicates whether to take a screenshot before starting a step.

with_screenshot_on_step_success(screenshot_on_step_success)

Accepts a Boolean argument, which indicates whether to take a screenshot after completing a step successfully.

with_screenshot_on_step_failure(screenshot_on_step_failure)

Accepts a Boolean argument, which indicates whether to take a screenshot after a step fails.

get_screenshot_on_step_start()

Returns whether to take a screenshot before starting a step.

get_screenshot_on_step_success()

Returns whether to take a screenshot after completing a step successfully.

get_screenshot_on_step_failure()

Returns whether to take a screenshot after a step fails.

disable_step_screenshots()

Disables all screenshot options (get_screenshot_on_step_start, get_screenshot_on_step_success, and get_screenshot_on_step_failure).

enable_step_screenshots()

Enables all screenshot options (get_screenshot_on_step_start, get_screenshot_on_step_success, and get_screenshot_on_step_failure). By default, all these methods are enabled.

setConfig(options) regarding CloudWatch metrics

For canaries using `syn-python-selenium-1.1` or later, the **(options)** for **setConfig** can include the following Boolean parameters that determine which metrics are published by the canary. The default for each of these options is `true`. The options that start with `aggregated` determine whether the metric is emitted without the `CanaryName` dimension. You can use these metrics to see the aggregated results for all of your canaries. The other options determine whether the metric is emitted with the `CanaryName` dimension. You can use these metrics to see results for each individual canary.

For a list of CloudWatch metrics emitted by canaries, see CloudWatch metrics published by canaries.

- `failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `failed_requests_metric` (boolean)— Whether to emit the `Failed requests` metric (with the `CanaryName` dimension) for this canary. The default is `true`.

- `2xx_metric` (boolean)— Whether to emit the `2xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `4xx_metric` (boolean)— Whether to emit the `4xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `5xx_metric` (boolean)— Whether to emit the `5xx` metric (with the `CanaryName` dimension) for this canary. The default is `true`.
- `step_duration_metric` (boolean)— Whether to emit the `Step duration` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `step_success_metric` (boolean)— Whether to emit the `Step success` metric (with the `CanaryName StepName` dimensions) for this canary. The default is `true`.
- `aggregated_failed_canary_metric` (boolean)— Whether to emit the `Failed` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_failed_requests_metric` (boolean)— Whether to emit the `Failed Requests` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_2xx_metric` (boolean)— Whether to emit the `2xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_4xx_metric` (boolean)— Whether to emit the `4xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.
- `aggregated_5xx_metric` (boolean)— Whether to emit the `5xx` metric (without the `CanaryName` dimension) for this canary. The default is `true`.

with_2xx_metric(2xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with the `CanaryName` dimension for this canary.

with_4xx_metric(4xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with the `CanaryName` dimension for this canary.

with_5xx_metric(5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with the `CanaryName` dimension for this canary.

withAggregated2xxMetric(aggregated2xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `2xx` metric with no dimension for this canary.

withAggregated4xxMetric(aggregated4xxMetric)

Accepts a Boolean argument, which specifies whether to emit a `4xx` metric with no dimension for this canary.

with_aggregated_5xx_metric(aggregated_5xx_metric)

Accepts a Boolean argument, which specifies whether to emit a `5xx` metric with no dimension for this canary.

with_aggregated_failed_canary_metric(aggregated_failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with no dimension for this canary.

with_aggregated_failed_requests_metric(aggregated_failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with no dimension for this canary.

with_failed_canary_metric(failed_canary_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed` metric with the `CanaryName` dimension for this canary.

with_failed_requests_metric(failed_requests_metric)

Accepts a Boolean argument, which specifies whether to emit a `Failed requests` metric with the `CanaryName` dimension for this canary.

with_step_duration_metric(step_duration_metric)

Accepts a Boolean argument, which specifies whether to emit a `Duration` metric with the `CanaryName` dimension for this canary.

with_step_success_metric(step_success_metric)

Accepts a Boolean argument, which specifies whether to emit a `StepSuccess` metric with the `CanaryName` dimension for this canary.

Methods to enable or disable metrics

disable_aggregated_request_metrics()

Disables the canary from emitting all request metrics that are emitted with no `CanaryName` dimension.

disable_request_metrics()

Disables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

disable_step_metrics()

Disables all step metrics, including both step success metrics and step duration metrics.

enable_aggregated_request_metrics()

Enables the canary to emit all request metrics that are emitted with no `CanaryName` dimension.

enable_request_metrics()

Enables all request metrics, including both per-canary metrics and metrics aggregated across all canaries.

enable_step_metrics()

Enables all step metrics, including both step success metrics and step duration metrics.

Usage in UI canaries

First, import the synthetics dependency and fetch the configuration. Then, set the configuration for each option by calling the setConfig method using one of the following options.

```
from aws_synthetics.common import synthetics_configuration

synthetics_configuration.set_config(
     {
        "screenshot_on_step_start": False,
        "screenshot_on_step_success": False,
        "screenshot_on_step_failure": True
     }
)
```

```
or
```

Or

```
synthetics_configuration.with_screenshot_on_step_start(False).with_screenshot_on_step_success(F
```

To disable all screenshots, use the disableStepScreenshots() function as in this example.

```
synthetics_configuration.disable_step_screenshots()
```

You can enable and disable screenshots at any point in the code. For example, to disable screenshots only for one step, disable them before running that step and then enable them after the step.

set_config(options) for UI canaries

Starting with `syn-python-selenium-1.1`, for UI canaries, `set_config` can include the following Boolean parameters:

- `continue_on_step_failure` (boolean)— Whether to continue with running the canary script after a step fails (this refers to the **executeStep** function). If any steps fail, the canary run will still be marked as failed. The default is `false`.

SyntheticsLogger class

`synthetics_logger` writes logs out to both the console and to a local log file at the same log level. This log file is written to both locations only if the log level is at or below the desired logging level of the log function that was called.

The logging statements in the local log file are prepended with "DEBUG: ", "INFO: ", and so on to match the log level of the function that was called.

Using `synthetics_logger` is not required to create a log file that is uploaded to your Amazon S3 results location. You could instead create a different log file in the `/tmp` folder. Any files created under the `/tmp` folder are uploaded to the results location in the S3 bucket as artifacts.

To use `synthetics_logger`:

```
from aws_synthetics.common import synthetics_logger
```

Useful function definitions:

Get log level:

```
log_level = synthetics_logger.get_level()
```

Set log level:

```
synthetics_logger.set_level()
```

Log a message with a specified level. The level can be `DEBUG`, `INFO`, `WARN`, or `ERROR`, as in the following syntax examples:

```
synthetics_logger.debug(message, *args, **kwargs)
```

```
synthetics_logger.info(message, *args, **kwargs)
```

```
synthetics_logger.log(message, *args, **kwargs)
```

```
synthetics_logger.warning(message, *args, **kwargs)
```

```
synthetics_logger.error(message, *args, **kwargs)
```

For information about debug parameters, see the standard Python documentation at [logging.debug]

In these logging functions, the `message` is the message format string. The `args` are the arguments that are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.
- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Examples:

Log a message with the level DEBUG:

```
synthetics_logger.debug('Starting step - login.')
```

Log a message with the level INFO. `logger.log` is a synonym for `logger.info`:

```
synthetics_logger.info('Successfully completed step - login.')
```

or

```
synthetics_logger.log('Successfully completed step - login.')
```

Log a message with the level WARN:

```
synthetics_logger.warning('Warning encountered trying to publish %s', 'CloudWatch
 Metric')
```

Log a message with the level ERROR:

```
synthetics_logger.error('Error encountered trying to publish %s', 'CloudWatch Metric')
```

Log an exception:

```
synthetics_logger.exception(message, *args, **kwargs)
```

Logs a message with level ERROR. Exception information is added to the logging message. You should call this function only from an exception handler.

For information about exception parameters, see the standard Python documentation at logging.exception

The `message` is the message format string. The `args` are the arguments, which are merged into `msg` using the string formatting operator.

There are three keyword arguments in `kwargs`:

- `exc_info`– If not evaluated as false, adds exception information to the logging message.

- `stack_info`– defaults to false. If true, adds stack information to the logging message, including the actual logging call.
- `extra`– The third optional keyword argument, which you can use to pass in a dictionary that is used to populate the `__dict__` of the `LogRecord` created for the logging event with user-defined attributes.

Example:

```
synthetics_logger.exception('Error encountered trying to publish %s', 'CloudWatch
 Metric')
```

Python and Selenium library classes and functions that apply to UI canaries only

The following CloudWatch Synthetics Selenium library functions for Python are useful only for UI canaries.

Topics

- SyntheticsBrowser class
- SyntheticsWebDriver class

SyntheticsBrowser class

> **Note**
>
> `SyntheticsBrowser` is supported only on the Chrome browser.

When you create a browser instance by calling `synthetics_webdriver.Chrome()`, the returned browser instance is of the type `SyntheticsBrowser`. The `SyntheticsBrowser` class inherits the WebDriver class and provides access to all the methods exposed by the WebDriver. It controls the ChromeDriver, and enables the canary script to drive the browser, allowing the Selenium WebDriver to work with Synthetics.

> **Note**
>
> Synthetics overrides the WebDriver `quit` method to take no action. You don't need to worry about quitting the browser, as Synthetics handles that for you.

In addition to the standard Selenium methods, it also provides the following methods.

Topics

- set_viewport_size(width, height)
- save_screenshot(filename, suffix)

set_viewport_size(width, height)

Sets the viewport of the browser. Example:

```
browser.set_viewport_size(1920, 1080)
```

save_screenshot(filename, suffix)

Saves screenshots to the `/tmp` directory. The screenshots are uploaded from there to the canary artifacts folder in the S3 bucket.

filename is the file name for the screenshot, and *suffix* is an optional string to be used for naming the screenshot.

Example:

```
browser.save_screenshot('loaded.png', 'page1')
```

SyntheticsWebDriver class

To use this class, use the following in your script:

```
from aws_synthetics.selenium import synthetics_webdriver
```

Topics

- add_execution_error(errorMessage, ex);
- add_user_agent(user_agent_str)
- execute_step(step_name, function_to_execute)
- get_http_response(url)
- Chrome()

add_execution_error(errorMessage, ex);

`errorMessage` describes the error and `ex` is the exception that is encountered

You can use `add_execution_error` to set execution errors for your canary. It fails the canary without interrupting the script execution. It also doesn't impact your `successPercent` metrics.

You should track errors as execution errors only if they are not important to indicate the success or failure of your canary script.

An example of the use of `add_execution_error` is the following. You are monitoring the availability of your endpoint and taking screenshots after the page has loaded. Because the failure of taking a screenshot doesn't determine availability of the endpoint, you can catch any errors encountered while taking screenshots and add them as execution errors. Your availability metrics will still indicate that the endpoint is up and running, but your canary status will be marked as failed. The following sample code block catches such an error and adds it as an execution error.

```
try:
    browser.save_screenshot("loaded.png")
except Exception as ex:
   self.add_execution_error("Unable to take screenshot", ex)
```

add_user_agent(user_agent_str)

Appends the value of `user_agent_str` to the browser's user agent header. You must assign `user_agent_str` before creating the browser instance.

Example:

```
await synthetics_webdriver.add_user_agent('MyApp-1.0')
```

`add_user_agent` should be used inside an `async` function.

execute_step(step_name, function_to_execute)

Processes one function. It also does the following:

- Logs that the step started.
- Takes a screenshot named `<stepName>-starting`.
- Starts a timer.

- Executes the provided function.
- If the function returns normally, it counts as passing. If the function throws, it counts as failing.
- Ends the timer.
- Logs whether the step passed or failed
- Takes a screenshot named `<stepName>-succeeded` or `<stepName>-failed`.
- Emits the `stepName` `SuccessPercent` metric, 100 for pass or 0 for failure.
- Emits the `stepName` `Duration` metric, with a value based on the step start and end times.
- Finally, returns what the `functionToExecute` returned or re-throws what `functionToExecute` threw.

Example:

```
from selenium.webdriver.common.by import By

def custom_actions():
        #verify contains
        browser.find_element(By.XPATH, "//*[@id=\"id_1\"][contains(text(),'login')]")
        #click a button
        browser.find_element(By.XPATH, '//*[@id="submit"]/a').click()

    await synthetics_webdriver.execute_step("verify_click", custom_actions)
```

get_http_response(url)

Makes an HTTP request to the provided URL and returns the response code of the HTTP request. If an exception occurred during the HTTP request, a string with value "error" is returned instead.

Example:

```
response_code = syn_webdriver.get_http_response(url)
if not response_code or response_code == "error" or response_code < 200 or
 response_code > 299:
    raise Exception("Failed to load page!")
```

Chrome()

Launches an instance of the Chromium browser and returns the created instance of the browser.

Example:

```
browser = synthetics_webdriver.Chrome()
browser.get("https://example.com/)
```

To launch a browser in incognito mode, use the following:

```
add_argument('--incognito')
```

To add proxy settings, use the following:

```
add_argument('--proxy-server=%s' % PROXY)
```

Example:

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("--incognito")
browser = syn_webdriver.Chrome(chrome_options=chrome_options)
```

Scheduling canary runs using cron

Using a cron expression gives you flexibility when you schedule a canary. Cron expressions contain five or six fields in the order listed in the following table. The fields are separated by spaces. The syntax differs depending on whether you are using the CloudWatch console to create the canary, or the AWS CLI or AWS SDKs. When you use the console, you specify only the first five fields. When you use the AWS CLI or AWS SDKs, you specify all six fields, and you must specify `*` for the `Year` field.

Field	Allowed values	Allowed special characters
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? L #

Field	Allowed values	Allowed special characters
Year	*	

Special characters

- The **,** (comma) includes multiple values in the expression for a field. For example, in the Month field, JAN,FEB,MAR would include January, February, and March.
- The **-** (dash) special character specifies ranges. In the Day field, 1-15 would include days 1 through 15 of the specified month.
- The ***** (asterisk) special character includes all values in the field. In the Hours field, ***** includes every hour. You cannot use ***** in both the Day-of-month and Day-of-week fields in the same expression. If you use it in one, you must use **?** in the other.
- The **/** (forward slash) specifies increments. In the Minutes field, you can enter 1/10 to specify every tenth minute, starting from the first minute of the hour (for example, the eleventh, twenty-first, and thirty-first minute, and so on).
- The **?** (question mark) specifies one or another. If you enter **7** in the Day-of-month field and you don't care what day of the week the seventh is, you can enter **?** in the Day-of-week field.
- The **L** wildcard in the Day-of-month or Day-of-week fields specifies the last day of the month or week.
- The **W** wildcard in the Day-of-month field specifies a weekday. In the Day-of-month field, **3W** specifies the weekday closest to the third day of the month.
- The **#** wildcard in the Day-of-week field specifies a certain instance of the specified day of the week within a month. For example, 3#2 is the second Tuesday of the month. The 3 refers to Tuesday because it is the third day of each week, and the 2 refers to the second day of that type within the month.

Limitations

- You can't specify the Day-of-month and Day-of-week fields in the same cron expression. If you specify a value or * (asterisk) in one of the fields, you must use a **?** (question mark) in the other.
- Cron expressions that lead to rates faster than one minute are not supported.
- You can't set a canary to wait for more than a year before running, so you can specify only * in the `Year` field.

Examples

You can refer to the following sample cron strings when you create a canary. The following examples are the correct syntax for using the AWS CLI or AWS SDKs to create or update a canary. If you are using the CloudWatch console, omit the final * in each example.

Expression	Meaning
`0 10 * * ? *`	Run at 10:00 am (UTC) every day
`15 12 * * ? *`	Run at 12:15 pm (UTC) every day
`0 18 ? * MON-FRI *`	Run at 6:00 pm (UTC) every Monday through Friday
`0 8 1 * ? *`	Run at 8:00 am (UTC) on the first day of each month
`0/10 * ? * MON-SAT *`	Run every 10 minutes Monday through Saturday of each week
`0/5 8-17 ? * MON-FRI *`	Run every five minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

Configuring your canary to retry automatically

When you create or update a canary, you can configure your canaries to automatically attempt additional runs when the scheduled one fails. This helps differentiate between genuine failures and temporary glitches, providing more reliable results. This feature is ideal for building more resilient monitoring systems while reducing false alarms and manual intervention.

To create a auto retry canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Canary**.
4. Under **Additional configuration**, **Auto-retry,** select the desired maximum retry number.

To update the maximum retry number for a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. You can do one of the following:

   - Select the canary and choose **Actions**, **Enable auto-retry**, and adjust the maximum retries.
   - Select the canary and choose **Actions**, **Edit**. In the **Edit detail** page, under **Additional configuration**, **Auto-retry**, adjust the retry configuration.

Limitations

Here are the limitations to configure auto retry.

- Supported only on runtime versions `syn-nodejs-puppeteer-10.0` or newer, `syn-nodejs-playwright-2.0` or newer, or `syn-python-selenium-5.1` or newer, or `syn-nodejs-3.0` or newer.
- Long running canaries which timeout after ten minutes are limited to one retry. All other canaries can support up to two retries.

Using dependencies with CloudWatch Synthetics canaries

This section explains how to use `Dependencies` in CloudWatch Synthetics canaries. The `Dependencies` field allows you to specify dependencies for your canaries, enabling you to include additional libraries or custom code that your canary scripts can use.

Overview

CloudWatch Synthetics canaries support specifying Lambda layers as dependencies. This feature allows you to:

- Share common code across multiple canaries
- Manage dependencies separately from your canary script code
- Reduce the size of your canary script by moving dependencies to a Lambda layer

Supported APIs

The `Dependencies` field is supported in the following APIs:

- CreateCanary
- UpdateCanary

- StartCanaryDryRun

Syntax

The `Dependencies` field is part of the code structure in the request syntax:

```
"Code": {
  "Handler": "string",
  "S3Bucket": "string",
  "S3Key": "string",
  "S3Version": "string",
  "ZipFile": blob,
  "Dependencies": [
    {
      "Type": "LambdaLayer",
      "Reference": "string"
    }
  ]
}
```

Using dependencies

Here are some examples and instructions for using the `Dependencies` field in different scenarios.

Creating a Canary with dependencies

When creating a canary, you can specify a Lambda layer as a dependency:

```
{
  "Name": "my-canary",
  "Code": {
    "Handler": "pageLoadBlueprint.handler",
    "S3Bucket": "my-bucket",
    "S3Key": "my-canary-script.zip",
    "Dependencies": [
      {
        "Type": "LambdaLayer",
        "Reference": "arn:aws:lambda:us-west-2:123456789012:layer:my-custom-layer:1"
      }
    ]
  },
```

```
  "ArtifactS3Location": "s3://my-bucket/artifacts/",
  "ExecutionRoleArn": "arn:aws:iam::123456789012:role/my-canary-role",
  "Schedule": {
    "Expression": "rate(5 minutes)"
  },
  "RuntimeVersion": "syn-nodejs-puppeteer-3.9"
}
```

Updating a Canary's dependencies

You can update a canary's dependencies using the UpdateCanary API:

```
{
  "Name": "my-canary",
  "Code": {
    "Dependencies": [
      {
        "Type": "LambdaLayer",
        "Reference": "arn:aws:lambda:us-west-2:123456789012:layer:my-updated-layer:2"
      }
    ]
  }
}
```

Removing dependencies

To remove dependencies from a canary, provide an empty array for the Dependencies field:

```
{
  "Name": "my-canary",
  "Code": {
    "Dependencies": []
  }
}
```

Testing dependencies with StartCanaryDryRun

Before updating a canary with new dependencies, you can test them using the StartCanaryDryRun API:

```
{
  "Name": "my-canary",
  "Code": {
    "Dependencies": [
      {
        "Type": "LambdaLayer",
        "Reference": "arn:aws:lambda:us-west-2:123456789012:layer:my-test-layer:3"
      }
    ]
  }
}
```

Limitations and considerations

- Only one Lambda layer can be specified as a dependency
- The role being used to create a canary with dependencies should have `lambda:GetLayerVersion` access to the dependency layer in addition to the necessary roles and permissions

Creating compatible Lambda layers

For information on how to create and package layers, see Managing Lambda dependencies with layers and to understand the packaging structure of a canary check based on the canary packing structure, see Writing a canary script.

Multilocation canaries

Amazon CloudWatch Synthetics lets you run the same canary across multiple AWS Regions simultaneously from a single point of management. These are called *multilocation canaries*. You create and manage the canary in one Region (the *primary location*), and CloudWatch Synthetics automatically replicates it to the additional Regions (the *replica locations*) that you choose. All runs, metrics, and artifacts are consolidated in the primary Region, giving you a unified view of your application's performance and availability worldwide.

Use multilocation canaries for the following benefits:

- **Make sure consistent user experience**—Run canaries from different parts of the world to verify that your users everywhere receive a consistent, high-quality experience.
- **Establish location-specific baselines**— Each Region might have different performance characteristics due to factors such as network latency and data center proximity. Running

canaries from multiple locations creates performance baselines for each location, helping you distinguish normal variations from anomalies.

- **Identify regional performance issues**— Performance and availability can vary significantly across locations due to network latency, ISP throttling, or regional outages. Testing from diverse global locations helps pinpoint Region-specific bottlenecks that might not be apparent from a single location.
- **Validate third-party services and CDNs**— Verify that your third-party dependencies such as payment processors, content delivery networks, and advertisement services are working across all Regions. Use the data to hold vendors accountable for their SLAs.
- **Reduce false positives**—Configure alarms that activate only when issues are detected from multiple locations. This approach avoids unnecessary notifications caused by isolated transient network glitches, so that your teams can focus on critical problems.

How multilocation canaries work

A multilocation canary uses a primary and replica model. The canary in the primary Region acts as the source of truth for configuration. CloudWatch Synthetics automatically creates replicas in your chosen Regions, and each replica runs independently using the same script, schedule, and environment variables. All mutating operations (create, update, start, stop, delete) are performed from the primary Region and asynchronously propagated to replicas. Run data from all locations is consolidated in the primary Region.

Prerequisites

Before you create a multilocation canary, consider the following requirements:

- Multilocation canaries require runtime version `syn-nodejs-puppeteer-16.0` or later, or `syn-nodejs-playwright-7.0` or later.
- Available in all AWS commercial Regions. Not available in AWS GovCloud (US) or China Regions.
- You can add up to 50 replica locations.
- Tags are not replicated to replica Regions. To add tags to a replica, navigate to the replica Region and add tags directly.
- Environment variables from the primary canary are applied to all replicas.
- All locations use the same script and schedule.
- Each replica requires its own VPC configuration if VPC connectivity is needed. VPC settings are not inherited from the primary Region.

- Cost scales linearly with the number of replicas. Each replica incurs the same cost as a standalone canary.

Topics

- Permissions for multilocation canaries
- Managing multilocation canaries
- Monitoring and troubleshooting multilocation canaries
- Using log centralization with multilocation canaries

Permissions for multilocation canaries

To create and manage multilocation canaries, the IAM principal must have the standard CloudWatch Synthetics permissions (see Required roles and permissions for CloudWatch canaries) plus the following:

- `synthetics:ReplicateCanary`—Allows the CloudWatch Synthetics service to create, update, and delete replicas in replica Regions on your behalf. If your policy already includes `synthetics:*`, this permission is included and no additional action is required. When you add, update, or remove replica locations, related activity appears in CloudTrail in each affected replica Region under the event name `ReplicateCanary`.

Condition keys for multilocation canaries

You can use condition keys in IAM policies to control which Regions can be used as replica locations. The following condition keys are available:

Condition key	Description	Type	Used with
`synthetics:AddReplicaLocations`	Filters access by the replica Regions specified in the request	ArrayOfString	`synthetics:CreateCanary` , `synthetics:UpdateCanary`
`synthetics:RemoveReplicaLocations`	Filters access by the replica Regions being removed in the request	ArrayOfString	`synthetics:UpdateCanary`

Example: Allow replication only to specific Regions

The following policy allows creating and updating canaries with replicas only in United States and Canada Regions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "synthetics:CreateCanary",
                "synthetics:UpdateCanary"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "synthetics:AddReplicaLocations": [
                        "us-*",
                        "ca-*"
                    ]
                }
            }
        }
    ]
}
```

Example: Deny replication to specific Regions

The following policy denies creating or updating canaries with replicas in `eu-west-1` or `ap-southeast-1`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "synthetics:CreateCanary",
                "synthetics:UpdateCanary"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "ForAnyValue:StringEquals": {
                    "synthetics:AddReplicaLocations": [
                        "eu-west-1",
                        "ap-southeast-1"
                    ]
                }
            }
        }
    ]
}
```

For more information about CloudWatch Synthetics permissions, see Required roles and permissions for CloudWatch canaries.

Managing multilocation canaries

Creating a multilocation canary

To create a multilocation canary, follow the steps in Creating a canary and select additional Regions under **Locations**. The current Region is automatically selected as the primary location.

When you create a multilocation canary, the following requirements apply:

- You can add up to 50 replica locations.
- The canary script .zip file must not exceed 50 MB.
- The canary name must not already exist in any of the selected replica Regions.
- Replica locations must be supported CloudWatch Synthetics commercial Regions. The primary Region cannot be selected as a replica location.
- Multilocation canaries cannot be associated with a group. If your canary belongs to a group, disassociate it from all groups before adding replica locations.
- If your canary requires VPC connectivity, you must configure VPC settings for each replica Region separately. VPC settings are not inherited from the primary.
- Tags are not replicated to replica Regions. To add tags to a replica, navigate to the replica Region and add tags directly.

After creation, replicas are provisioned asynchronously. You can monitor the replication progress on the canary detail page in the **Configuration** tab. When a replica reaches `InSync` state, it is fully provisioned and running. You can also view the replica canary directly in the replica Region. For more information, see Replication status.

If a replica fails to provision, the primary canary is still created successfully. The failed replica displays an `Inconsistent` replication status. You can retry by updating the canary.

Adding or removing replica locations

To add or remove replica locations, edit the canary from the primary Region and update the selections under **Locations**. You cannot edit a multilocation canary from a replica Region. For more information about editing a canary, see Edit or delete a canary.

When you add or remove replica locations, the following requirements apply:

- You cannot modify a multilocation canary while replication is in progress. Wait for all replicas to reach `InSync` or `Inconsistent` state before making changes. For more information, see Replication status.
- When you add replica locations, you must include the canary code in your request.
- When you update a canary that has inconsistent replicas, you must include the canary code, unless you are removing all inconsistent replicas in the same request.
- You cannot add and remove the same location in a single request.
- You cannot remove a replica location that does not exist.

Starting and stopping a multilocation canary

When you start or stop a multilocation canary, the action applies to the primary canary and all replica locations.

When you start or stop a multilocation canary, the following requirements apply:

- You must start or stop the canary from the primary Region. Start and stop operations are not permitted from a replica Region.
- You cannot start or stop a multilocation canary while replication is in progress. Wait for all replicas to reach `InSync` or `Inconsistent` state before starting or stopping. For more information, see Replication status.
- The canary must be in the `READY` or `STOPPED` state to start, or in the `RUNNING` state to stop.

Deleting a multilocation canary

To delete a multilocation canary, you must first remove all replica locations. For more information, see Edit or delete a canary. The following requirements apply:

- You must delete the canary from the primary Region. Delete operations are not permitted from a replica Region.
- Wait for all replicas to be fully removed before you attempt to delete. You can monitor removal progress in the **Configuration** tab. For more information, see Replication status.
- The canary must be in the `STOPPED`, `READY`, or `ERROR` state. If the canary is running, stop it first.

Replication status

Replication status indicates the synchronization state of a replica with the primary canary. You can view the replication status on the canary detail page in the **Configuration** tab.

Replicas can have the following replication status values:

- **InProgress**—The replica is being created or updated. Changes from the primary are being propagated to the replica.
- **InSync**—The replica is fully synchronized with the primary canary configuration.
- **Inconsistent**—The replica is out of sync with the primary canary. This can occur if a propagation failed. You can retry by updating the canary again.

The overall *ReplicationState* on the primary canary shows the aggregate state across all replicas. If any replica is `InProgress`, the overall state is `InProgress`. If any replica is `Inconsistent`, the overall state is `Inconsistent`.

Monitoring and troubleshooting multilocation canaries

When you create a multilocation canary, CloudWatch Synthetics consolidates run data, artifacts, and key metrics in the primary Region, giving you a unified view of your canary's performance across all locations. If the primary Region becomes unavailable, you can still access run data, metrics, and logs directly in each replica Region.

Canary runs

In the CloudWatch Synthetics console in the primary Region, you can view canary runs from the primary and all replica Regions in a single table on the canary detail page. Each run includes a **Location** column that shows the Region where it was executed. You can filter runs by location to compare performance and identify issues in specific Regions.

You can also navigate to the CloudWatch Synthetics console in any replica Region to view runs that were executed in that Region independently.

Artifacts

Screenshots, HAR files, and reports from all locations are stored in the same Amazon S3 bucket configured for your canary. Replicas write artifacts directly to the primary Region's Amazon S3 bucket under a regional prefix, so that you can identify which location generated them:

- Primary runs: `s3://bucket/canary/primaryRegion/canaryName/year/month/day/hour/minute-second-millisecond`
- Replica runs: `s3://bucket/canary/primaryRegion/canaryName/year/month/day/hour/replicaRegion/minute-second-millisecond`

Because Amazon S3 buckets are globally accessible, you can retrieve artifacts from any Region. When you select a run on the canary detail page, the CloudWatch Synthetics console displays the associated screenshots and reports inline, regardless of which location executed the run.

Metrics

Multilocation canaries publish the following metrics to CloudWatch in the primary Region:

- **SuccessPercent**—Published with a `Location` dimension for each Region where the canary runs. An aggregate `SuccessPercent` metric without the `Location` dimension represents the combined success rate across all locations. To monitor overall canary health across all locations, create a CloudWatch alarm on this aggregate metric.
- **Duration**—Published with a `Location` dimension for each Region where the canary runs.

All canary metrics are also published in the respective replica Region. If the primary Region is unavailable, you can navigate to the CloudWatch console in any replica Region to view metrics for that location.

For a complete list of metrics published by canaries, see CloudWatch metrics published by canaries.

Logs

When you select a run on the canary detail page, you can view the associated logs for that run regardless of which location executed it. This allows you to debug failures from any location directly in the primary Region.

To centralize canary logs from all Regions into a single CloudWatch Logs view, you can configure cross-Region log centralization. For more information, see Using log centralization with multilocation canaries.

Using log centralization with multilocation canaries

When you configure a canary to run from multiple locations, each execution produces logs in its respective Region. You can use CloudWatch Logs centralization to aggregate these logs into a single Region, making it easier to monitor and troubleshoot canary results from one place.

For information about how log centralization works, including concepts, permissions, and setup, see Centralize logging from multiple accounts and Regions in the *CloudWatch Logs User Guide*.

Configuring log centralization for a multilocation canary

To centralize logs from a multilocation canary, create a log centralization rule in CloudWatch Logs with the following recommended settings.

Setting	Recommended value
Source regions	Same Regions as the replica locations on your multilocation canary
Destination account	Same account where your multilocation canary lives
Destination region	Primary Region of your multilocation canary
Filter	`LogGroupName ^ /aws/lambda/cwsyn-`
Destination log group	`${source.logGroup}`

This rule centralizes all canary logs from the selected source Regions to the destination Region—including logs from non-multilocation canaries running in those Regions. For multilocation canaries specifically, using `${source.logGroup}` as the destination log group places replica logs in the same log group as the primary canary logs, because primary and replica canaries share the same log group name.

For instructions on creating a log centralization rule, see Centralize logging from multiple accounts and Regions.

Viewing centralized logs

If you use the recommended settings above, logs from all canary execution Regions appear in the primary Region's log group. If you configured a different destination, your logs appear in the destination you specified.

To view centralized logs (console)

1. Open the CloudWatch console.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose your multilocation canary name to open its details page.
4. In the **Selected canary runs** table, choose a canary run that occurred in the primary Region.
5. Choose the **Logs** tab at the bottom of the canary details page.
6. Choose **View/Download in Logs Insights**.

You can use CloudWatch Logs Insights to query across all log entries. For example, to find failures from a specific Region:

```
fields @timestamp, @message, @logStream
| filter @aws.region like /us-west-2/
| filter @message like /ERROR/
```

Groups

You can create *groups* to associate canaries with each other, including cross-Region canaries. Using groups can help you with managing and automating your canaries, and you can also view aggregated run results and statistics for all canaries in a group.

Groups are global resources. When you create a group, it is replicated across all AWS Regions that support groups, and you can add canaries from any of these Regions to it, and view it in any of these Regions. Although the group ARN format reflects the Region name where it was created, a group is not constrained to any Region. This means that you can put canaries from multiple Regions into the same group, and then use that group to view and manage all of those canaries in a single view.

Groups are supported in all Regions except the Regions that are disabled by default. For more information about these Regions, see Enabling a Region .

Each group can contain as many as 10 canaries. You can have as many as 20 groups in your account. Any single canary can be a member of up to 10 groups.

To create a group

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose **Create Group**.
4. Under **Group Name**, enter a name for the group.
5. Select canaries to associate with this group. To select a canary, type its complete name in **Exact canary name** and choose **Search**. Then select the check box next to the canary name. If there are multiple canaries with the same name in different Regions, be sure to select the canaries that you want.

   You can repeat this step to associate as many as 10 canaries with the group.
6. (Optional) Under **Tags**, add one or more key-value pairs as tags for this group. Tags can help you identify and organize your AWS resources and track your AWS costs. For more information, see Tagging your Amazon CloudWatch resources.
7. Choose **Create Group**.

Test a canary locally

This section explains how to modify, test, and debug CloudWatch Synthetics canaries directly within the Microsoft Visual Studio code editor or the JetBrains IDE code editor. The local debugging environment uses a Serverless Application Model (SAM) container to simulate a Lambda function to emulate the behavior of a Synthetics canary.

> **Note**
>
> It is impractical to perform locally debug canaries that rely on visual monitoring. Visual monitoring rely on capturing base screenshots during an initial run, and then comparing these screenshots to the screenshots from subsequent runs. In a local development environment, runs are not stored or tracked, and each iteration is an independent, standalone run. The absence of a canary run history makes it impractical to debug canaries that rely on visual monitoring.

Prerequisites

1. Choose or create an Amazon S3 bucket to use to store artifacts from local canary test runs, such as HAR files and screenshots. This requires you to be provisioned with IAM. If you skip setting up Amazon S3 buckets you can still test your canary locally, but you will see an error message about the missing bucket and you won't have access to canary artifacts.

   If you use an Amazon S3 bucket, we recommend that you set the bucket lifecycle to delete objects after a few days, to save costs. For more information, see Managing your storage lifecycle.
2. Set up a default AWS profile for your AWS account. For more information, see Configuration and credential file settings.
3. Set the debug environment's default AWS Region to your preferred Region, such as `us-west-2`.
4. Install the AWS SAM CLI. For more information, see Installing the AWS SAM CLI.
5. Install Visual Studio Code Editor or JetBrains IDE. For more information, see Visual Studio Code or JetBrains IDE
6. Install Docker to work with the AWS SAM CLI. Make sure to start the docker daemon. For more information, see Installing Docker to use with the AWS SAM CLI.

   Alternatively, You can install other container management software such as Rancher, as long as it uses the Docker runtime.
7. Install an AWS toolkit extension for your preferred editor. For more information, see Installing the AWS Toolkit for Visual Studio Code or Installing the AWS Toolkit for JetBrains.

Topics

- Set up the testing and debugging environment
- Use Visual Studio Code IDE
- Use JetBrains IDE
- Run a canary locally with the SAM CLI
- Integrate your local testing environment into an existing canary package
- Change the CloudWatch Synthetics runtime
- Common errors

Set up the testing and debugging environment

First, clone the Github repository that AWS provides by entering the following command. The repository contains code samples for both Node.js canaries and Python canaries.

```
git clone https://github.com/aws-samples/synthetics-canary-local-debugging-sample.git
```

Then do one of the following, depending on the language of your canaries.

For Node.js canaries

1. Go to the Node.js canary source directory by entering the following command.

   ```
   cd synthetics-canary-local-debugging-sample/nodejs-canary/src
   ```

2. Enter the following command to install canary dependencies.

   ```
   npm install
   ```

For Python canaries

1. Go to the Python canary source directory by entering the following command.

   ```
   cd synthetics-canary-local-debugging-sample/python-canary/src
   ```

2. Enter the following command to install canary dependencies.

   ```
   pip3 install -r requirements.txt -t .
   ```

Use Visual Studio Code IDE

The Visual Studio launch configuration file is located at `.vscode/launch.json`. It contains configurations to allow the template file to be discovered by Visual Studio code. It defines a Lambda payload with required parameters to invoke the canary successfully. Here's the launch configuration for a Node.js canary:

```
{
            ...
```

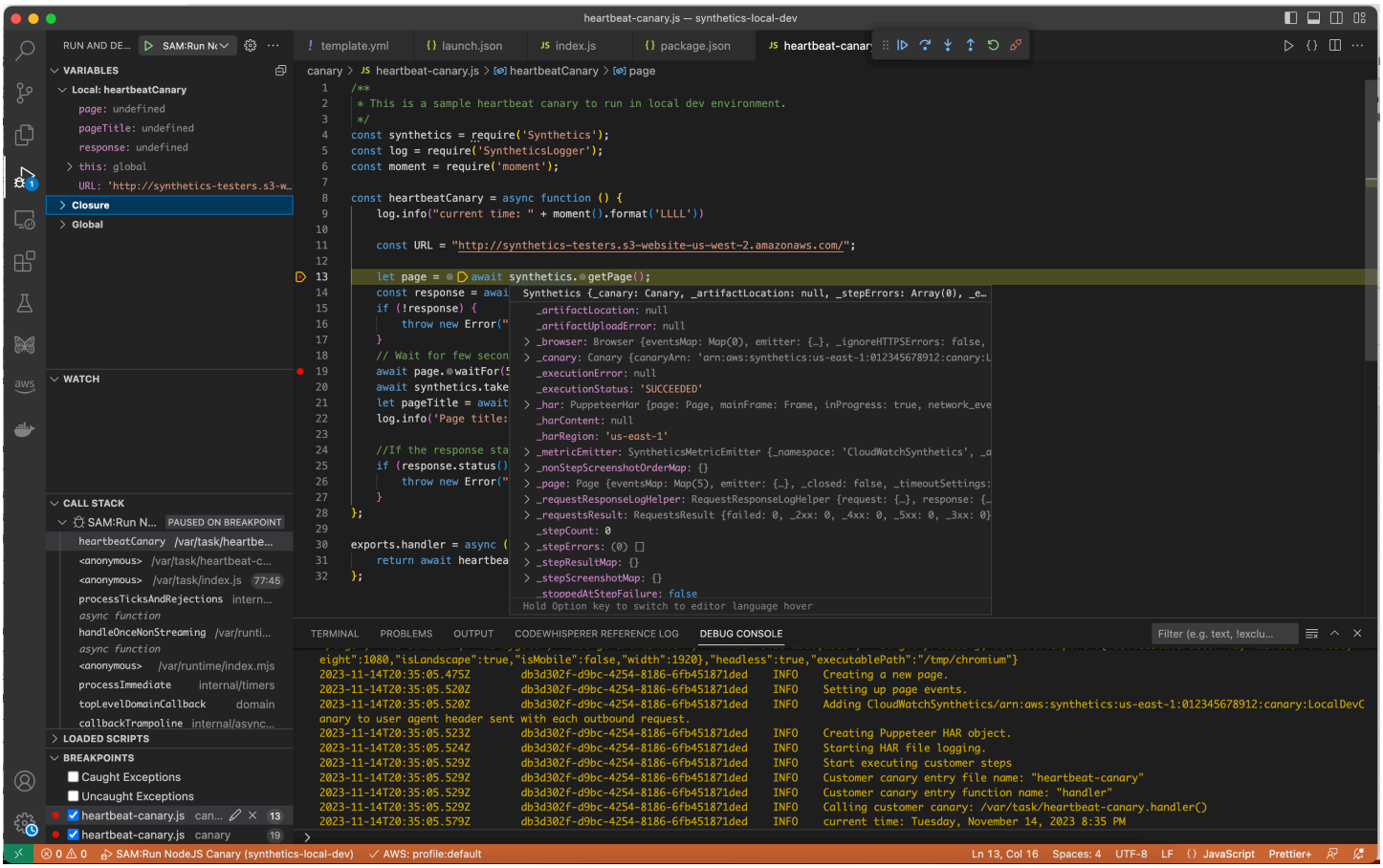
```
            ...
            "lambda": {
                "payload": {
                    "json": {
                        // Canary name. Provide any name you like.
                        "canaryName": "LocalSyntheticsCanary",
                        // Canary artifact location
                        "artifactS3Location": {
                            "s3Bucket": "cw-syn-results-123456789012-us-west-2",
                            "s3Key": "local-run-artifacts",
                        },
                        // Your canary handler name
                        "customerCanaryHandlerName": "heartbeat-canary.handler"
                    }
                },
                // Environment variables to pass to the canary code
                "environmentVariables": {}
            }
        }
    ]
}
```

You can also optionally provide the following fields in the payload JSON:

- `s3EncryptionMode` Valid values: `SSE_S3` | `SSE_KMS`
- `s3KmsKeyArn` Valid value: *`KMS Key ARN`*
- `activeTracing` Valid values: `true` | `false`
- `canaryRunId` Valid value: *`UUID`* This parameter is required if active tracing is enabled.

To debug the canary in Visual Studio, add breakpoints in the canary code where you want to pause execution. To add a breakpoint, choose the editor margin and go to **Run and Debug** mode in the editor. Run the canary by clicking on the play button. When the canary runs, the logs will be tailed in the debug console, providing you with real-time insights into the canary's behavior. If you added breakpoints, the canary execution will pause at each breakpoint, allowing you to step through code and inspect variable values, instance methods, object attributes, and the function call stack.

There is no cost incurred for running and debugging canaries locally, except for the artifacts stored in the Amazon S3 bucket and the CloudWatch metrics generated by each local run.



Use JetBrains IDE

After you have the AWS Toolkit for JetBrains extension installed, be sure that the Node.js plugin and JavaScript debugger are enabled to run, if you are debugging a Node.js canary. Then follow these steps.

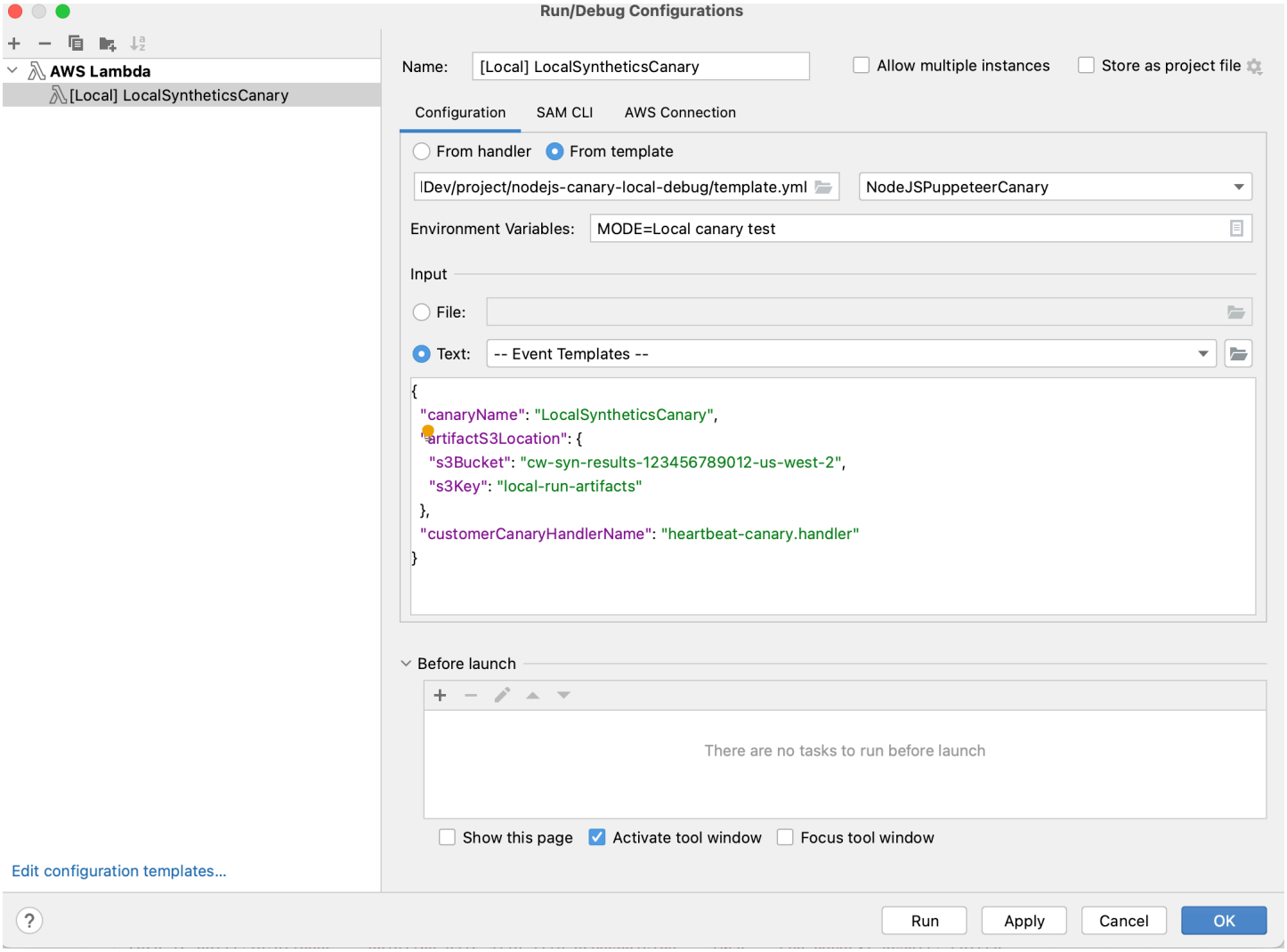
Debug a canary using JetBrains IDE

1. In the left navigation pane of JetBrains IDE, choose **Lambda**, then choose the local configuration template.
2. Enter a name for the run configuration, such as **`LocalSyntheticsCanary`**
3. Choose **From template**, choose the file browser in the template field, then select the **template.yml** file from the project, either from the **nodejs** directory or the **python** directory.
4. In the **Input** section, enter the payload for the canary as shown in the following screen.

```
{
 "canaryName": "LocalSyntheticsCanary",
 "artifactS3Location": {
```

```
    "s3Bucket": "cw-syn-results-123456789012-us-west-2",
    "s3Key": "local-run-artifacts"
 },
 "customerCanaryHandlerName": "heartbeat-canary.handler"
}
```

You can also set other environment variables in the payload JSON, as listed in Use Visual Studio Code IDE.



Run a canary locally with the SAM CLI

Use the one of the following procedures to run your canary locally using the Serverless Application Model (SAM) CLI. Be sure to specify your own Amazon S3 bucket name for `s3Bucket` in `event.json`

To use the SAM CLI to run a Node.js canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/nodejs-canary
```

2. Enter the following commands.

```
sam build
sam local invoke -e ../event.json
```

To use the SAM CLI to run a Python canary

1. Go to the source directory by entering the following command.

```
cd synthetics-canary-local-debugging-sample/python-canary
```

2. Enter the following commands.

```
sam build
sam local invoke -e ../event.json
```

Integrate your local testing environment into an existing canary package

You can integrate local canary debugging into your existing canary package by copying three files:

- Copy the `template.yml` file into your canary package root. Be sure to modify the path for `CodeUri` to point to the directory where your canary code exists.
- If you're working with a Node.js canary, copy the `cw-synthetics.js` file to your canary source directory. If you're working with a Python canary, copy the `cw-synthetics.py` to your canary source directory.
- Copy the launch configuration file `.vscode/launch.json` into the package root. Make sure to put it inside the `.vscode` directory; create it if it doesn't exist already.

Change the CloudWatch Synthetics runtime

As part of your debugging, you might want to try running a canary with a different CloudWatch Synthetics runtime, instead of the latest runtime. To do so, find the runtime that you want to use from one of the following tables. Be sure to select the runtime for the correct Region. Then paste the ARN for that runtime into the appropriate place in your `template.yml` file, and then run the canary.

Node.js and Puppeteer runtimes

ARNs for syn-nodejs-puppeteer-18.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-18.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:84`
	Firefox	`arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:25`
US East (Ohio)	Chrome	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:87`
	Firefox	`arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24`
US West (N. California)	Chrome	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:85`
	Firefox	`arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24`
US West (Oregon)	Chrome	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:87`

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
	Chrome	arn:aws:lambda:us-northeast-1:127750070752:layer:Synthetics:33
	Firefox	arn:aws:lambda:us-northeast-1:127750070752:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:85
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:85
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:48
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:85

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:88
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:71
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:85
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:60
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:89

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:84
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:67
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:58
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:41
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:41

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:25
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:32
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:85
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:116
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:84
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:85

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:85
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:59
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:85
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:86
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:60

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:86
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:84
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:85
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:57
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
	Chrome	arn:aws:lambda:me-west-1:833660969287:layer:Synthetics:16

Region	Browser	ARN
	Firefox	arn:aws:lambda:me-west-1:833660969287:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:12
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:33
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:86
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:80
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:81
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:24

ARNs for syn-nodejs-puppeteer-17.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-17.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:82
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:23
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:85
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:83
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:85
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:83
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:83

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:58
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:65
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:39
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:56
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:83

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:39
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:23
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:69
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:86
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:87
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:46
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:30
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:83
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:83
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:114

Region	Browser	ARN
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:82
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:83
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:83
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:84
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:84
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:83
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:58
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:83
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:57

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:55
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:31
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:50
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:84
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:78

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:79
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22

ARNs for syn-nodejs-puppeteer-16.1

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-16.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:81
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:84
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:84
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:82
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:82
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:57
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:64

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:38
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:55
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:82
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:38
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:22
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:68

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:85
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:86
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:81
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:45
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:29

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:82
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:82
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:113
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:81
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:82
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:83
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:83
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:57
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:82
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:56
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:54
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:83
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:77
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:78
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21

ARNs for syn-nodejs-puppeteer-16.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-16.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:80

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:83
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:81
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:83
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:81
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:81

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:56
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:63
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:37
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:54
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:81

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:37
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:21
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:67
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:84
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:85
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:80

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:44
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:28
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:81
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:81
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:112

Region	Browser	ARN
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:80
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:81
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:82
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:80

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:82
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:56
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:55

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:53
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:29
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:82
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:76
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:77

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20

ARNs for syn-nodejs-puppeteer-15.1

The following table lists the ARNs to use for version syn-nodejs-puppeteer-15.1 of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:79
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:82
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:80
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:82

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:80
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:80
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:55
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:62
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:36

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:53
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:80
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:36
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:20
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:66
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:83

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:84
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:79
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:43
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:27
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:80

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:80
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:111
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:79
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:80
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:80

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:79
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:81
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:80
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:55

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:80
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:54
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:52
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:28
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:81

Region	Browser	ARN
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:75
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:76
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:19

ARNs for syn-nodejs-puppeteer-15.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-15.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:75
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:78

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:76
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:78
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:76
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:76
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:51

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:58
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:32
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:49
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:76
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:32

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:16
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:62
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:79
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:80
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:75
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:39

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:23
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:76
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:76
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:107
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:75

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:76
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:76
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:77
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:75
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:77

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:76
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:51
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:76
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:50
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:48

Region	Browser	ARN
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:24
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:77
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:71
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:72
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15

ARNs for syn-nodejs-puppeteer-14.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-14.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:74
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:77
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:75
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:77
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:75
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:75

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:50
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:57
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:31
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:48
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:75

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:31
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:15
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:61
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:78
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:79
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:74

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:38
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:22
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:75
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:75
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:106

Region	Browser	ARN
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:74
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:75
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:75
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:76
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:74

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:76
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:75
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:50
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:75
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:49

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:47
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:23
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:74
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:50
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:76

Region	Browser	ARN
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:70
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:71
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14

ARNs for syn-nodejs-puppeteer-13.1

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-13.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:73
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:76

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:74
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:76
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:74
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:74
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:49

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:56
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:30
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:47
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:74
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:14
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:60
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:77
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:78
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:73
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:37

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:21
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:74
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:74
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:105
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:73

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:74
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:74
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:75
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:73
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:75

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:74
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:49
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:74
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:48
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:46

Region	Browser	ARN
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:22
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:73
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:49
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:75
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:69

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:70
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:13

ARNs for syn-nodejs-puppeteer-13.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-13.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:71
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:74
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:72

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:74
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:72
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:72
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:47
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:54

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:28
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:45
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:72
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:28
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:12
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:58

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:75
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:76
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:71
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:35
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:72
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:72
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:103
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:71
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:72

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:72
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:73
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:71
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:73
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:72

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:47
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:72
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:46
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:44
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:20

Region	Browser	ARN
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:71
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:47
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:73
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:67
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:68

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:11

ARNs for syn-nodejs-puppeteer-12.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-12.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:67
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:70
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:68
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:70

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:68
	Firefox	arn:aws:lambda:af-south-1:490290609897:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:68
	Firefox	arn:aws:lambda:ap-east-1:865491513039:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:43
	Firefox	arn:aws:lambda:ap-south-2:378461160691:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:50
	Firefox	arn:aws:lambda:ap-southeast-3:490290609897:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:24

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-5:654654156625:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:41
	Firefox	arn:aws:lambda:ap-southeast-4:491585149102:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:68
	Firefox	arn:aws:lambda:ap-south-1:565803892007:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:24
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:8
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:54
	Firefox	arn:aws:lambda:ap-northeast-3:865491513039:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:71

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-2:296224022648:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:72
	Firefox	arn:aws:lambda:ap-southeast-1:960282847306:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:67
	Firefox	arn:aws:lambda:ap-southeast-2:635750524965:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:31
	Firefox	arn:aws:lambda:ap-east-2:654654156625:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:15
	Firefox	arn:aws:lambda:ap-southeast-7:654654156625:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:68

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-1:213485871058:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:68
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:99
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:67
	Firefox	arn:aws-cn:lambda:cn-north-1:453652488670:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:68
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:68

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:69
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:67
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:69
	Firefox	arn:aws:lambda:eu-south-1:282507446575:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:68
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:43

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-2:378461160691:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:68
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:42
	Firefox	arn:aws:lambda:eu-central-2:378461160691:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:40
	Firefox	arn:aws:lambda:il-central-1:378461160691:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:16
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:67

Region	Browser	ARN
	Firefox	arn:aws:lambda:me-south-1:491585149102:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:43
	Firefox	arn:aws:lambda:me-central-1:491585149102:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:69
	Firefox	arn:aws:lambda:sa-east-1:756306329178:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:63
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:488211338238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:64
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:488211338238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:7

ARNs for syn-nodejs-puppeteer-11.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-11.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:63
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:66
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:64
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:66
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:64
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:64

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:39
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:46
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:20
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:37
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:64

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:20
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:4
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:50
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:67
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:68
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:63

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:27
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:11
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:64
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:64
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:95

Region	Browser	ARN
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:63
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:64
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:64
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:65
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:63

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:65
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:64
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:39
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:64
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:38

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:36
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:12
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:63
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:39
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:65

Region	Browser	ARN
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:59
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:60
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPuppeteer-Firefox:3

ARNs for syn-nodejs-puppeteer-10.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-10.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:58
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:61
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:59
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:61

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:59
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:59
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:34
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:41
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:15
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:32
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:59
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:18
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:45
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:62
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:63
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:58

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:24
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:6
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:59
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:59
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:90
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:58
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:59
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:59
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:60
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:58
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:60
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:59

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:34
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:59
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:33
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:31
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:58
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:34
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:60
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:54
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:55

ARNs for syn-nodejs-puppeteer-9.1

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-9.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:53
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:56
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:54
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:56
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:54
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:54
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:29
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:36
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:10
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:27
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:54
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:17

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:40
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:57
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:58
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:53
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:25
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:54
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:54
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:85
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:54
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:55
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:54

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:55
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:53
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:55
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:54
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:29
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:54
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:28
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:26
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:53
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:29
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:55

Region	ARN
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:50`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:51`

ARNs for syn-nodejs-puppeteer-9.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-9.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:51`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:54`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:52`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:54`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:52`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:52`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:27`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:34`

Region	ARN
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:8
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:25
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:52
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:16
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:38
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:55
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:56
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:51
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:26
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:52
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:52
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:83

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:52
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:53
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:52
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:53
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:51
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:53
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:52
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:27
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:52
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:26
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:24
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:51

Region	ARN
Middle East (UAE)	`arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:27`
South America (São Paulo)	`arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:53`
AWS GovCloud (US-East)	`arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:48`
AWS GovCloud (US-West)	`arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:49`

ARNs for syn-nodejs-puppeteer-8.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-8.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:48`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:50`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:48`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:51`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:48`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:49`

Region	ARN
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:24
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:30
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:22
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:48
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics:15
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:34
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:51
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:53
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:48
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics:23
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:48
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:48

Region	ARN
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:80
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:49
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:50
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:48
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:50
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:48
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:49
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:48
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:24
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:48
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:23
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:21

Region	ARN
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:48
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:23
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:49
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:45
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:46

ARNs for syn-nodejs-puppeteer-7.0

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-7.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:44
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:46
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:44
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:47
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:44

Region	ARN
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:45
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:20
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:26
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics:7
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:44
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:46
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:49
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:44
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:44

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:76
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:45
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:46
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:44
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:46
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:44
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:45
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:44
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:44
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:19

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:17
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics:4
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:44
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:19
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:45
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:41
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:42

ARNs for syn-nodejs-puppeteer-6.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-6.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:41
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:43
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:41

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:44
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:41
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:42
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:41
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:46
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:41
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:41

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:41
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:42
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:41
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:43
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:41
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:42
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:41
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:41
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:41
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:42
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:39
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:39

ARNs for syn-nodejs-puppeteer-5.2

The following table lists the ARNs to use for version `syn-nodejs-puppeteer-5.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:42
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:44
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:42
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:45

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:42
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:43
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:18
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:24
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:16
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:42
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:28
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:47
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:42
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:42
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:42

Region	ARN
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:74
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:44
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:42
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:44
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:42
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:43
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:42
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:42
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:17
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:15

Region	ARN
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:42
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:17
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:43
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:40
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:40

Node.js and Playwright runtimes

ARNs for syn-nodejs-playwright-9.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-9.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
	Chrome	arn:aws:lambda:us-northeast-1:127750070752:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:us-northeast-1:127750070752:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24

Region	Browser	ARN
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:32
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24

Region	Browser	ARN
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:31
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:25
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:29

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24

Region	Browser	ARN
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24

Region	Browser	ARN
	Chrome	arn:aws:lambda:me-west-1:833660969287:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:me-west-1:833660969287:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:11
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:31
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:30
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:29

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:24

ARNs for syn-nodejs-playwright-8.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-8.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22

Region	Browser	ARN
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:28

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:23

Region	Browser	ARN
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:30

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22

Region	Browser	ARN
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:28

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22

Region	Browser	ARN
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:14
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:28

Region	Browser	ARN
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22

ARNs for syn-nodejs-playwright-7.1

The following table lists the ARNs to use for version `syn-nodejs-playwright-7.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21

Region	Browser	ARN
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:27

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21

Region	Browser	ARN
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:22
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:27

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:29
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21

Region	Browser	ARN
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:27

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21

Region	Browser	ARN
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:28

Region	Browser	ARN
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21

ARNs for syn-nodejs-playwright-7.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-7.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:26

Region	Browser	ARN
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20

Region	Browser	ARN
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:21
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:26

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:28
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20

Region	Browser	ARN
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:25

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20

Region	Browser	ARN
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:26

Region	Browser	ARN
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20

ARNs for syn-nodejs-playwright-6.1

The following table lists the ARNs to use for version `syn-nodejs-playwright-6.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	`arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:25`
	Firefox	`arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19`
US East (Ohio)	Chrome	`arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:25`
	Firefox	`arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19`
US West (N. California)	Chrome	`arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:25`
	Firefox	`arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19`
US West (Oregon)	Chrome	`arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:25`
	Firefox	`arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19`

Region	Browser	ARN
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:25

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:20
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19

Region	Browser	ARN
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:27
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:24

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:24
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19

Region	Browser	ARN
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:24
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:25

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19

Region	Browser	ARN
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:26
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:25
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:24
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:24

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:19

ARNs for syn-nodejs-playwright-6.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-6.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15

Region	Browser	ARN
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:21

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:22
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:16

Region	Browser	ARN
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:23

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15

Region	Browser	ARN
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:21

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15

Region	Browser	ARN
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:22
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:20

Region	Browser	ARN
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:15

ARNs for syn-nodejs-playwright-5.1

The following table lists the ARNs to use for version `syn-nodejs-playwright-5.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13

Region	Browser	ARN
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13

Region	Browser	ARN
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:14
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:21
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13

Region	Browser	ARN
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13

Region	Browser	ARN
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13

ARNs for syn-nodejs-playwright-5.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-5.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

Region	Browser	ARN
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:18

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

Region	Browser	ARN
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:19
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:13
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:18

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:20
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:17
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

Region	Browser	ARN
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:17
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:17
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:18

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

Region	Browser	ARN
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:19

Region	Browser	ARN
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:18
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:17
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

Region	Browser	ARN
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:17
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:12

ARNs for syn-nodejs-playwright-4.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-4.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:13

Region	Browser	ARN
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7

Region	Browser	ARN
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:14

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:8
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7

Region	Browser	ARN
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:15
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:12
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:13

Region	Browser	ARN
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:12
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:11
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:6
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7

Region	Browser	ARN
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:13

Region	Browser	ARN
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:14
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7

Region	Browser	ARN
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:13
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:12
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:12
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:7

ARNs for syn-nodejs-playwright-3.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-3.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	Browser	ARN
US East (N. Virginia)	Chrome	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
US East (Ohio)	Chrome	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
US West (N. California)	Chrome	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
US West (Oregon)	Chrome	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Africa (Cape Town)	Chrome	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:7

Region	Browser	ARN
	Firefox	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Hong Kong)	Chrome	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Hyderabad)	Chrome	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Jakarta)	Chrome	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Malaysia)	Chrome	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1

Region	Browser	ARN
Asia Pacific (Melbourne)	Chrome	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Mumbai)	Chrome	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (New Zealand)	Chrome	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:5
	Firefox	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:2
Asia Pacific (Osaka)	Chrome	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Seoul)	Chrome	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:7

Region	Browser	ARN
	Firefox	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Singapore)	Chrome	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Sydney)	Chrome	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Taipei)	Chrome	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Asia Pacific (Thailand)	Chrome	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:6
	Firefox	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1

Region	Browser	ARN
Asia Pacific (Tokyo)	Chrome	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Canada (Central)	Chrome	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Canada West (Calgary)	Chrome	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
China (Beijing)	Chrome	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:6
	Firefox	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
China (Ningxia)	Chrome	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:6

Region	Browser	ARN
	Firefox	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Frankfurt)	Chrome	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Ireland)	Chrome	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (London)	Chrome	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Milan)	Chrome	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1

Region	Browser	ARN
Europe (Paris)	Chrome	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Spain)	Chrome	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Stockholm)	Chrome	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Europe (Zurich)	Chrome	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Israel (Tel Aviv)	Chrome	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:7

Region	Browser	ARN
	Firefox	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Mexico (Central)	Chrome	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:8
	Firefox	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Middle East (Bahrain)	Chrome	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
Middle East (UAE)	Chrome	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1
South America (São Paulo)	Chrome	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
	Firefox	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:1

Region	Browser	ARN
AWS GovCloud (US-East)	Chrome	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:8
	Firefox	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:3
AWS GovCloud (US-West)	Chrome	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:8
	Firefox	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright-Firefox:3

ARNs for syn-nodejs-playwright-2.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-2.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
US West (N. Californi a)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:4

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:7
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:4

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:9
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:4

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:4
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:3

ARNs for syn-nodejs-playwright-1.0

The following table lists the ARNs to use for version `syn-nodejs-playwright-1.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJsPlaywright:8

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJsPlaywright:10
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJsPlaywright:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJsPlaywright:1
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJsPlaywright:1

Node.js runtimes

ARNs for syn-nodejs-5.3

The following table lists the ARNs to use for version `syn-nodejs-5.3` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:20
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:20
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:20
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:20
	arn:aws:lambda:us-northeast-1:127750070752:layer:AWS-CW-SyntheticsNodeJs:20
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:20

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:20
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:20

Region	ARN
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:20
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:20
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:20
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:20
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:20
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:20
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:20

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:20
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:20
	arn:aws:lambda:me-west-1:833660969287:layer:AWS-CW-SyntheticsNodeJs:10
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:20
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:20
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:20
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:20

ARNs for syn-nodejs-5.2

The following table lists the ARNs to use for version `syn-nodejs-5.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:18
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:18
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:18

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:18
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:18

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:18
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:18
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:18
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:18
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:18
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:18
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:18

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:18
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:18
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:18
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:18
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJs:10
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:18
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:18
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:18

ARNs for syn-nodejs-5.1

The following table lists the ARNs to use for version `syn-nodejs-5.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:17
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:17
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:17
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:17
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:17

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:17
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:17
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:17
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:17
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:17
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:17

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:17
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:17
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:17
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:17
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:17
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:17
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:17
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:17

ARNs for syn-nodejs-5.0

The following table lists the ARNs to use for version `syn-nodejs-5.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:16`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:16`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:16`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:16`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:16`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:16`

Region	ARN
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:16
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:16
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:16
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:16
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:16
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:16

Region	ARN
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:16
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:16
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:16
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:16
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:16
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:16

Region	ARN
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:16

ARNs for syn-nodejs-4.2

The following table lists the ARNs to use for version `syn-nodejs-4.2` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:15
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:15
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:15
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:15
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:15

Region	ARN
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:15
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:15
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:15
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:15

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:15
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:15
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:15
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:15
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:15
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:15

Region	ARN
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:15
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:15
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:15

ARNs for syn-nodejs-4.1

The following table lists the ARNs to use for version `syn-nodejs-4.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:12
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:12
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:12
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:12
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:12

Region	ARN
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:12
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:12

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:12
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:12
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:12
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:12
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:12
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:12

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:12
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:12
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:12
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:12
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:12

ARNs for syn-nodejs-4.0

The following table lists the ARNs to use for version `syn-nodejs-4.0` of the CloudWatch Synthetics runtimein each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:11
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:11
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:11
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:11
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:11

Region	ARN
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:11

Region	ARN
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:11
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:11
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:11
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:11
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:11
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:11
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:11

Region	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:11
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:11
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:11
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:11
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:11
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:11
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:11

ARNs for syn-nodejs-3.1

The following table lists the ARNs to use for version `syn-nodejs-3.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:9
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:9
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:9

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:9
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:9

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:9
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:9
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:9
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:9
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:9
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:9
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:9

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:9
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:9
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:9
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:9
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJs:9
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJs:9
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:9
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:9
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:9

ARNs for syn-nodejs-3.0

The following table lists the ARNs to use for version `syn-nodejs-3.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:AWS-CW-SyntheticsNodeJs:1
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:AWS-CW-SyntheticsNodeJs:1
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:AWS-CW-SyntheticsNodeJs:1
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:AWS-CW-SyntheticsNodeJs:1
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:AWS-CW-SyntheticsNodeJs:1

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:AWS-CW-SyntheticsNodeJs:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:AWS-CW-SyntheticsNodeJs:1
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:AWS-CW-SyntheticsNodeJs:1
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:AWS-CW-SyntheticsNodeJs:1
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:AWS-CW-SyntheticsNodeJs:1
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:AWS-CW-SyntheticsNodeJs:1

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:AWS-CW-SyntheticsNodeJs:1
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:AWS-CW-SyntheticsNodeJs:1
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:AWS-CW-SyntheticsNodeJs:1
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:AWS-CW-SyntheticsNodeJs:1
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:AWS-CW-SyntheticsNodeJs:1
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:AWS-CW-SyntheticsNodeJs:1
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:AWS-CW-SyntheticsNodeJs:1
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:AWS-CW-SyntheticsNodeJs:1

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:AWS-CW-SyntheticsNodeJs:1
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:AWS-CW-SyntheticsNodeJs:1

Python and Selenium runtimes

ARNs for syn-python-selenium-13.0

The following table lists the ARNs to use for version `syn-python-selenium-13.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:71
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:74
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:72
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:73
	arn:aws:lambda:us-northeast-1:127750070752:layer:Synthetics_Selenium:33
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:72
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:71

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:34
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:72
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:75
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:70
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:72
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:59
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:76
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:71
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:66
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:57
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:41
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:47

Region	ARN
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:32
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:72
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:115
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:70
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:70
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:72
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:58
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:72
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:73
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:59
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:73
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:71

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:72
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:56
	arn:aws:lambda:me-west-1:833660969287:layer:Synthetics_Selenium:10
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:33
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:73
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:68
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:69

ARNs for syn-python-selenium-12.0

The following table lists the ARNs to use for version `syn-python-selenium-12.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:69
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:72
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:70

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:71
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:70
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:69
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:57
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:64
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:39
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:55
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:70
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:45
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:68
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:73
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:74

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:69
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:32
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:30
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:70
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:70
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:113
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:68
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:68
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:70
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:71
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:69
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:71

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:70
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:57
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:70
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:56
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:54
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:31
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:49
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:71
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:66
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:67

ARNs for syn-python-selenium-11.1

The following table lists the ARNs to use for version `syn-python-selenium-11.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:68
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:71
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:69
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:70
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:69
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:68
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:56
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:63
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:38
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:54
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:69
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:44

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:67
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:72
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:73
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:68
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:31
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:29
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:69
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:69
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:112
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:67
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:67
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:69

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:70
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:68
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:70
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:69
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:56
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:69
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:55
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:53
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:30
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:70
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:65
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:66

ARNs for syn-python-selenium-11.0

The following table lists the ARNs to use for version `syn-python-selenium-11.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	`arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:66`
US East (Ohio)	`arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:69`
US West (N. California)	`arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:67`
US West (Oregon)	`arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:68`
Africa (Cape Town)	`arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:67`
Asia Pacific (Hong Kong)	`arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:66`
Asia Pacific (Hyderabad)	`arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:54`
Asia Pacific (Jakarta)	`arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:61`
Asia Pacific (Malaysia)	`arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:36`
Asia Pacific (Melbourne)	`arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:52`
Asia Pacific (Mumbai)	`arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:67`

Region	ARN
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:42
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:65
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:70
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:71
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:66
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:29
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:27
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:67
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:67
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:110
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:65
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:65

Region	ARN
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:67
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:68
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:66
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:68
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:67
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:54
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:67
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:53
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:51
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:28
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:68
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:63

Region	ARN
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:64

ARNs for syn-python-selenium-10.0

The following table lists the ARNs to use for version `syn-python-selenium-10.0` of the CloudWatch Synthetics runtimein each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:62
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:65
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:63
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:64
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:63
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:62
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:50
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:57
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:32

Region	ARN
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:48
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:63
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:38
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:61
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:66
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:67
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:62
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:25
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:23
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:63
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:63
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:106

Region	ARN
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:61
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:61
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:63
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:64
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:62
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:64
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:63
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:50
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:63
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:49
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:47
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:24

Region	ARN
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:64
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:59
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:60

ARNs for syn-python-selenium-9.0

The following table lists the ARNs to use for version `syn-python-selenium-9.0` of the CloudWatch Synthetics runtimein each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:61
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:64
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:62
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:63
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:62
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:61
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:49

Region	ARN
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:56
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:31
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:47
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:62
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:37
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:60
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:65
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:66
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:61
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:24
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:22
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:62

Region	ARN
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:62
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:105
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:60
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:60
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:62
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:63
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:61
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:63
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:62
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:49
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:62
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:48

Region	ARN
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:46
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:23
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:61
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:49
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:63
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:58
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:59

ARNs for syn-python-selenium-8.0

The following table lists the ARNs to use for version `syn-python-selenium-8.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:57
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:60
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:58

Region	ARN
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:59
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:58
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:57
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:45
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:52
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:27
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:43
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:58
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:33
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:56
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:61
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:62

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:57
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:20
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:18
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:58
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:58
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:101
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:56
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:56
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:58
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:59
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:57
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:59

Region	ARN
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:58
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:45
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:58
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:44
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:42
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:19
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:57
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:45
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:59
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:54
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:55

ARNs for syn-python-selenium-7.0

The following table lists the ARNs to use for version `syn-python-selenium-7.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:49
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:52
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:50
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:51
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:50
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:49
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:37
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:44
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:19
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:35
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:50
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:25

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:48
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:53
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:54
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:49
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:12
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:10
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:50
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:50
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:93
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:48
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:48
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:50

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:51
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:49
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:51
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:50
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:37
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:50
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:36
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:34
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:11
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:49
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:37
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:51

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:46
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:47

ARNs for syn-python-selenium-6.0

The following table lists the ARNs to use for version `syn-python-selenium-6.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:47
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:50
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:48
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:49
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:48
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:47
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:35
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:42

Region	ARN
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:17
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:33
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:48
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:25
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:46
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:51
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:52
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:47
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:10
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:8
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:48
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:48

Region	ARN
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:91
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:46
China (Ningxia)	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:46
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:48
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:49
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:47
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:49
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:48
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:35
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:48
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:34
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:32

Region	ARN
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:9
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:47
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:35
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:49
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:44
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:45

ARNs for syn-python-selenium-5.1

The following table lists the ARNs to use for version `syn-python-selenium-5.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:45
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:48
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:46
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:47

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:46
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:45
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:33
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:40
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:15
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:31
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:46
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:24
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:44
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:49
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:50
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:45

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:8
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:6
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:46
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:89
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:44
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:44
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:46
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:47
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:45
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:47
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:46

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:33
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:46
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:32
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:30
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:7
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:45
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:33
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:47
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:42
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:43

ARNs for syn-python-selenium-5.0

The following table lists the ARNs to use for version `syn-python-selenium-5.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:43
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:46
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:44
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:45
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:44
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:43
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:31
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:38
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:13
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:29
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:44
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:23

Region	ARN
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:42
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:47
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:48
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:43
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:3
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:4
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:44
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:44
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:87
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:43
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:43
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:44

Region	ARN
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:45
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:43
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:45
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:44
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:31
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:44
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:30
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:28
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:5
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:43
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:31
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:45

Region	ARN
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:41
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:42

ARNs for syn-python-selenium-4.1

The following table lists the ARNs to use for version `syn-python-selenium-4.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:40
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:43
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:41
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:42
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:41
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:40
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:28
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:35

Region	ARN
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:10
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:26
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:41
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:22
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:39
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:44
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:45
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:40
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:4
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:1
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:41
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:41

Region	ARN
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:84
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:40
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:40
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:41
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:42
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:40
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:42
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:41
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:28
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:41
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:27
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:25

Region	ARN
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:3
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:40
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:28
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:42
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:38
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:39

ARNs for syn-python-selenium-4.0

The following table lists the ARNs to use for version `syn-python-selenium-4.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:38
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:41
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:39
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:40

Region	ARN
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:39
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:38
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:26
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:33
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:8
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:24
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:39
Asia Pacific (New Zealand)	arn:aws:lambda:ap-southeast-6:779414629109:layer:Synthetics_Selenium:21
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:37
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:42
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:43
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:38

Region	ARN
Asia Pacific (Taipei)	arn:aws:lambda:ap-east-2:471112897378:layer:Synthetics_Selenium:5
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:39
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:39
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:82
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:38
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:38
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:39
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:40
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:38
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:40
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:39
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:26

Region	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:39
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:25
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:23
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:38
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:26
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:40
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:36
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:37

ARNs for syn-python-selenium-3.0

The following table lists the ARNs to use for version `syn-python-selenium-3.0` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics_Selenium:32
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics_Selenium:34

Region	ARN
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics_Selenium:32
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics_Selenium:34
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics_Selenium:32
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics_Selenium:32
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics_Selenium:20
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics_Selenium:26
Asia Pacific (Malaysia)	arn:aws:lambda:ap-southeast-5:035872523913:layer:Synthetics_Selenium:7
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics_Selenium:18
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics_Selenium:32
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics_Selenium:30
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics_Selenium:34
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics_Selenium:37

Region	ARN
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics_Selenium:32
Asia Pacific (Thailand)	arn:aws:lambda:ap-southeast-7:851725245975:layer:Synthetics_Selenium:3
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics_Selenium:32
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics_Selenium:32
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics_Selenium:76
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics_Selenium:32
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics_Selenium:32
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics_Selenium:32
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics_Selenium:34
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics_Selenium:32
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics_Selenium:33
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics_Selenium:32

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics_Selenium:20
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics_Selenium:32
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics_Selenium:19
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics_Selenium:17
Mexico (Central)	arn:aws:lambda:mx-central-1:654654265476:layer:Synthetics_Selenium:4
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics_Selenium:32
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics_Selenium:19
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics_Selenium:33
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics_Selenium:30
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics_Selenium:31

ARNs for syn-python-selenium-2.1

The following table lists the ARNs to use for version `syn-python-selenium-2.1` of the CloudWatch Synthetics runtime in each AWS Region where it is available.

Region	ARN
US East (N. Virginia)	arn:aws:lambda:us-east-1:378653112637:layer:Synthetics:29
US East (Ohio)	arn:aws:lambda:us-east-2:772927465453:layer:Synthetics:31
US West (N. California)	arn:aws:lambda:us-west-1:332033056316:layer:Synthetics:29
US West (Oregon)	arn:aws:lambda:us-west-2:760325925879:layer:Synthetics:31
Africa (Cape Town)	arn:aws:lambda:af-south-1:461844272066:layer:Synthetics:29
Asia Pacific (Hong Kong)	arn:aws:lambda:ap-east-1:129828061636:layer:Synthetics:29
Asia Pacific (Hyderabad)	arn:aws:lambda:ap-south-2:280298676434:layer:Synthetics:17
Asia Pacific (Jakarta)	arn:aws:lambda:ap-southeast-3:246953257743:layer:Synthetics:23
Asia Pacific (Melbourne)	arn:aws:lambda:ap-southeast-4:200724813040:layer:Synthetics:15
Asia Pacific (Mumbai)	arn:aws:lambda:ap-south-1:724929286329:layer:Synthetics:29
Asia Pacific (Osaka)	arn:aws:lambda:ap-northeast-3:608016332111:layer:Synthetics:27
Asia Pacific (Seoul)	arn:aws:lambda:ap-northeast-2:989515803484:layer:Synthetics:30

Region	ARN
Asia Pacific (Singapore)	arn:aws:lambda:ap-southeast-1:068035103298:layer:Synthetics:34
Asia Pacific (Sydney)	arn:aws:lambda:ap-southeast-2:584677157514:layer:Synthetics:29
Asia Pacific (Tokyo)	arn:aws:lambda:ap-northeast-1:172291836251:layer:Synthetics:29
Canada (Central)	arn:aws:lambda:ca-central-1:236629016841:layer:Synthetics:29
Canada West (Calgary)	arn:aws:lambda:ca-west-1:944448206667:layer:Synthetics:73
China (Beijing)	arn:aws-cn:lambda:cn-north-1:422629156088:layer:Synthetics:29
China (Ningxia);	arn:aws-cn:lambda:cn-northwest-1:474974519687:layer:Synthetics:29
Europe (Frankfurt)	arn:aws:lambda:eu-central-1:122305336817:layer:Synthetics:29
Europe (Ireland)	arn:aws:lambda:eu-west-1:563204233543:layer:Synthetics:31
Europe (London)	arn:aws:lambda:eu-west-2:565831452869:layer:Synthetics:29
Europe (Milan)	arn:aws:lambda:eu-south-1:525618516618:layer:Synthetics:30
Europe (Paris)	arn:aws:lambda:eu-west-3:469466506258:layer:Synthetics:29

Region	ARN
Europe (Spain)	arn:aws:lambda:eu-south-2:029793053121:layer:Synthetics:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:162938142733:layer:Synthetics:29
Europe (Zurich)	arn:aws:lambda:eu-central-2:224218992030:layer:Synthetics:16
Israel (Tel Aviv)	arn:aws:lambda:il-central-1:313249807427:layer:Synthetics:14
Middle East (Bahrain)	arn:aws:lambda:me-south-1:823195537320:layer:Synthetics:29
Middle East (UAE)	arn:aws:lambda:me-central-1:239544149032:layer:Synthetics:16
South America (São Paulo)	arn:aws:lambda:sa-east-1:783765544751:layer:Synthetics:30
AWS GovCloud (US-East)	arn:aws-us-gov:lambda:us-gov-east-1:946759330430:layer:Synthetics:29
AWS GovCloud (US-West)	arn:aws-us-gov:lambda:us-gov-west-1:946807836238:layer:Synthetics:29

Common errors

Error: Running AWS SAM projects locally requires Docker. Have you got it installed and running?

Make sure to start Docker on your computer.

SAM local invoke failed: An error occurred (ExpiredTokenException) when calling the GetLayerVersion operation: The security token included in the request is expired

Make sure that the AWS default profile is set up.

More common errors

For more information about common errors with the SAM, see AWS SAM CLI troubleshooting .

Troubleshooting a failed canary

If your canary fails, check the following for troubleshooting.

General troubleshooting

- Use the canary details page to find more information. In the CloudWatch console, choose **Canaries** in the navigation pane and then choose the name of the canary to open the canary details page. In the **Availability** tab, check the **SuccessPercent** metric to see whether the problem is constant or intermittent.

  While still in the **Availability** tab, choose a failed data point to see screenshots, logs, and step reports (if available) for that failed run.

  If a step report is available because steps are part of your script, check to see which step has failed and see the associated screenshots to see the issue that your customers are seeing.

  You can also check the HAR files to see if one or more requests are failing. You can dig deeper by using logs to drill down on failed requests and errors. Finally, you can compare these artifacts with the artifacts from a successful canary run to pinpoint the issue.

  By default, CloudWatch Synthetics captures screenshots for each step in a UI canary. However, your script might be configured to disable screenshots. During debugging, you may want to enable screenshots again. Similarly, for API canaries you might want to see HTTP request and response headers and body during debugging. For information about how to include this data in the report, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).
- If you had a recent deployment to your application, roll it back and then debug later.
- Connect to your endpoint manually to see if you can reproduce the same issue.

Topics

- Canary fails after Lambda environment update
- My canary is blocked by AWS WAF
- Waiting for an element to appear
- Node is either not visible or not an HTMLElement for page.click()

- Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied
- Error: Protocol error (Runtime.callFunctionOn): Target closed.
- Canary Failed. Error: No datapoint - Canary Shows timeout error
- Trying to access an internal endpoint
- Canary runtime version upgrade and downgrade issues
- Cross-origin request sharing (CORS) issue
- Canary race condition issues
- Troubleshooting a canary on a VPC
- Troubleshooting an auto retry canary

Canary fails after Lambda environment update

CloudWatch Synthetics canaries are implemented as Lambda functions in your account. These Lambda functions are subject to regular Lambda runtime updates containing security updates, bug fixes, and other improvements. Lambda strives to provide runtime updates that are backward-compatible with existing functions. However, as with software patching, there are rare cases in which a runtime update can negatively impact an existing function. If you believe your canary has been impacted by a Lambda runtime update, you can use the Lambda runtime management manual mode (in supported Regions) to temporarily roll back the Lambda runtime version. This keeps your canary function working and minimizes disruption, providing time to remedy the incompatibility before returning to the latest runtime version.

If your canary is failing after a Lambda runtime update, the best solution is to upgrade to one of the newest Synthetics runtimes. For more information about the latest runtimes, see Synthetics runtime versions.

As an alternative solution, in Regions where Lambda runtime management controls are available, you can revert a canary back to an older Lambda managed runtime, using manual mode for runtime management controls. You can set manual mode using either the AWS CLI or by using the Lambda console, using the steps below in the following sections.

> **Warning**
>
> When you change the runtime settings to manual mode, your Lambda function won't receive automatic security updates until it is reverted back to Auto mode. During this period, your Lambda function might be susceptible to security vulnerabilities.

Prerequisites

- Install jq
- Install the latest version of the AWS CLI. For more information, see AWS CLI install and update instructions .

Step 1: Obtain the Lambda function ARN

Run the following command to retrieve the `EngineArn` field from the response. This `EngineArn` is the ARN of the Lambda function that is associated with the canary. You will use this ARN in the following steps.

```
aws synthetics get-canary --name my-canary | jq '.Canary.EngineArn'
```

Example output of `EngingArn`:

```
"arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-dc5015c2-db17-4cb5-
afb1-EXAMPLE991:8"
```

Step 2: Obtain the last good Lambda runtime version ARN

To help understand whether your canary was impacted by a Lambda runtime update, check whether the date and time when the Lambda runtime version ARN changes in your logs appeared to the date and time when you saw impact to your canary. If they do not match, it is probably not a Lambda runtime update that is causing your issues.

If your canary is impacted by a Lambda runtime update, you must identify the ARN of the working Lambda runtime version that you were previously using. Follow the instructions in Identifying runtime version changes to find the ARN of the previous runtime. Record the runtime version ARN, and continue to Step 3. for setting the runtime management configuration.

If your canary has not yet been impacted by a Lambda environment update, then you can find the ARN of the Lambda runtime version that you are currently using. Run the following command to retrieve the `RuntimeVersionArn` of the Lambda function from the response.

```
aws lambda get-function-configuration \
--function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991:8" | jq '.RuntimeVersionConfig.RuntimeVersionArn'
```

Example output of `RuntimeVersionArn`:

```
"arn:aws:lambda:us-
west-2::runtime:EXAMPLE647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

Step 3: Updating the Lambda runtime management configuration

You can use either the AWS CLI or the Lambda console to update the runtime management configuration.

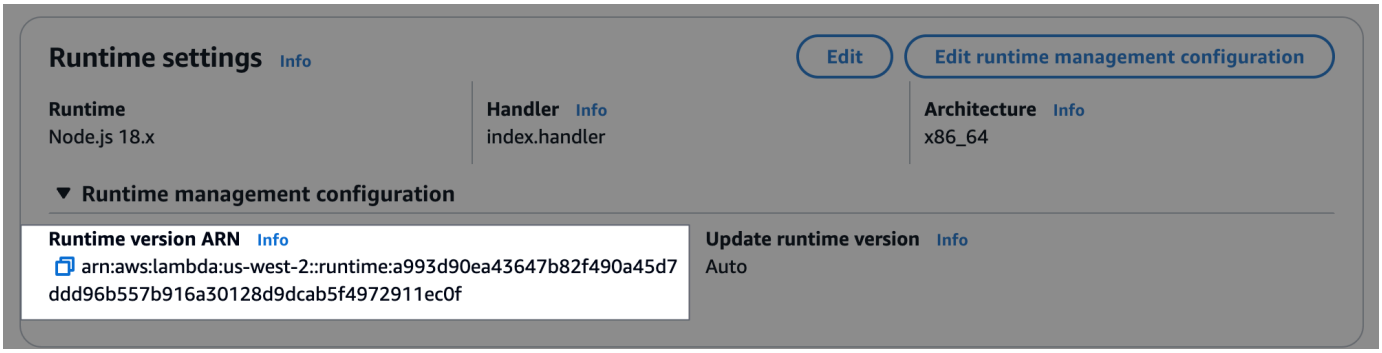
To set Lambda runtime management configuration manual mode using the AWS CLI

Enter the following command to change the runtime management of the Lambda function to manual mode. Be sure to replace the *`function-name`* and *`qualifier`* with the Lambda function ARN and Lambda function version number respectively, using the values you found in Step 1. Also replace the the *`runtime-version-arn`* with the version ARN that you found in Step 2.

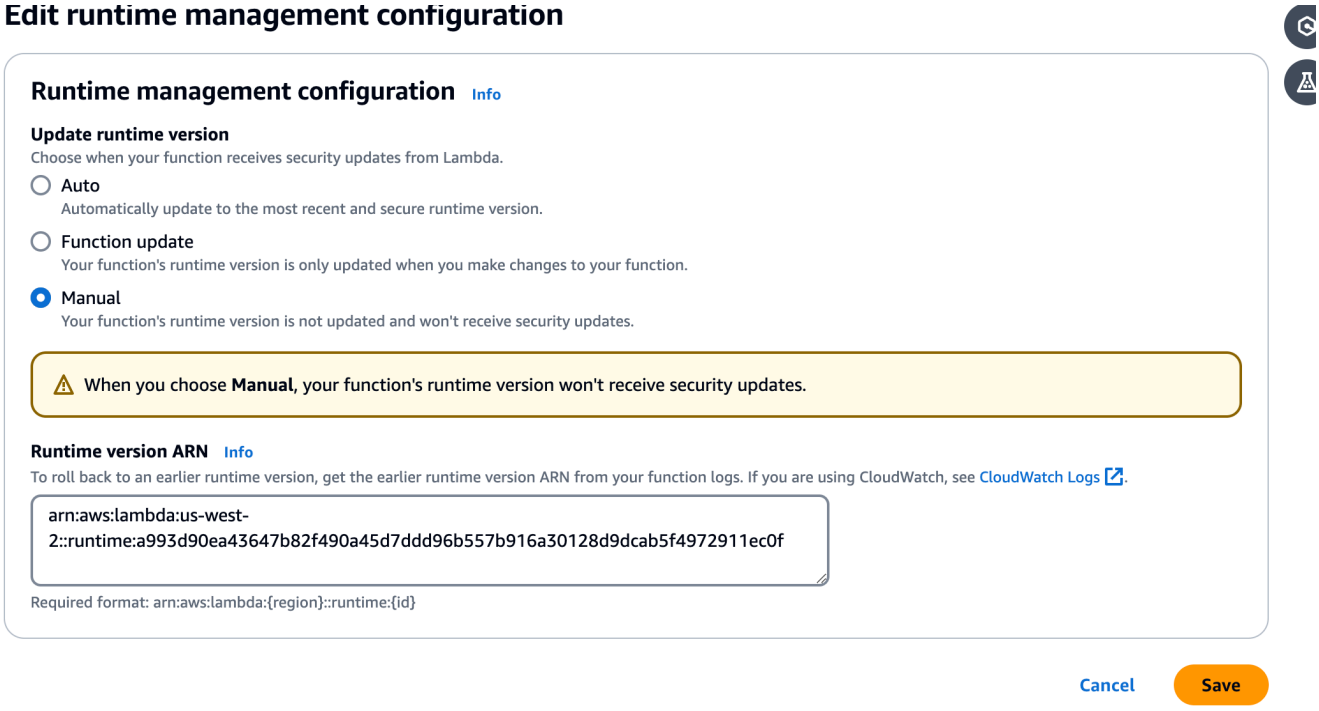
```
aws lambda put-runtime-management-config \
    --function-name "arn:aws:lambda:us-west-2:123456789012:function:cwsyn-my-canary-
dc5015c2-db17-4cb5-afb1-EXAMPLE991" \
    --qualifier 8 \
    --update-runtime-on "Manual" \
    --runtime-version-arn "arn:aws:lambda:us-
west-2::runtime:a993d90ea43647b82f490a45d7ddd96b557b916a30128d9dcab5f4972911ec0f"
```

To change a canary to manual mode using the Lambda console

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose the **Versions** tab, choose the version number link that corresponds to your ARN, and choose the **Code** tab.
3. Scroll down to **Runtime settings**, expand **Runtime management configuration**, and copy the the **Runtime version ARN** .



4. Choose **Edit runtime management configuration**, choose **Manual**, paste the runtime version ARN that you copied earlier into the **Runtime version ARN** field. Then choose **Save**.



My canary is blocked by AWS WAF

To allow canary traffic through AWS WAF,create a AWS WAF string match condition that allows a custom string that you specify. For more information, see Working with string match conditions in the AWS WAF documentation.

It is strongly recommend that you use your own custom user-agent string instead of using default values. This provides better control over AWS WAF filtering and improves security.

To set a custom user-agent string, do the following:

- For Playwright runtimes, you can append your AWS WAF approved custom user-agent string using the Synthetics configuration file. For more information, see CloudWatch Synthetics configurations.
- For Puppeteer or Selenium runtimes, you can add your custom user-agent string using supported library functions. For Puppeteer runtimes, see async addUserAgent(page, userAgentString);. For Selenium runtimes, see add_user_agent(user_agent_str).

Waiting for an element to appear

After analyzing your logs and screenshots, if you see that your script is waiting for an element to appear on screen and times out, check the relevant screenshot to see if the element appears on the page. Verify your `xpath` to make sure that it is correct.

For Puppetteer-related issues, check Puppeteer's GitHub page or internet forums.

Node is either not visible or not an HTMLElement for page.click()

If a node is not visible or is not an `HTMLElement` for `page.click()`, first verify the `xpath` that you are using to click the element. Also, if your element is at the bottom of the screen, adjust your viewport. CloudWatch Synthetics by default uses a viewport of 1920 * 1080. You can set a different viewport when you launch the browser or by using the Puppeteer function `page.setViewport`.

Unable to upload artifacts to S3, Exception: Unable to fetch S3 bucket location: Access Denied

If your canary fails because of an Amazon S3 error, CloudWatch Synthetics was unable to upload screenshots, logs, or reports created for the canary because of permission issues. Check the following:

- Check that the canary's IAM role has the `s3:ListAllMyBuckets` permission, the `s3:GetBucketLocation` permission for the correct Amazon S3 bucket, and the `s3:PutObject` permission for the bucket where the canary stores its artifacts. If the canary performs visual monitoring, the role also needs the `s3:GetObject` permission for the bucket. These same permissions are also required in the Amazon VPC S3 Gateway Endpoint Policy, if the canary is deployed in a VPC with a VPC endpoint.
- If the canary uses an AWS KMS customer managed key for encryption instead of the standard AWS managed key (default), the canary's IAM role might not have the permission to encrypt or decrypt using that key. For more information, see Encrypting canary artifacts.
- Your bucket policy might not allow the encryption mechanism that the canary uses. For example, if your bucket policy mandates to use a specific encryption mechanism or KMS key, then you must select the same encryption mode for your canary.

If the canary performs visual monitoring, see Updating artifact location and encryption when using visual monitoring for more information.

Error: Protocol error (Runtime.callFunctionOn): Target closed.

This error appears if there are some network requests after the page or browser is closed. You might have forgotten to wait for an asynchronous operation. After executing your script, CloudWatch Synthetics closes the browser. The execution of any asynchronous operation after the browser is closed might cause `target closed error`.

Canary Failed. Error: No datapoint - Canary Shows timeout error

This means that your canary run exceeded the timeout. The canary execution stopped before CloudWatch Synthetics could publish success percent CloudWatch metrics or update artifacts such as HAR files, logs and screenshots. If your timeout is too low, you can increase it.

By default, a canary timeout value is equal to its frequency. You can manually adjust the timeout value to be less than or equal to the canary frequency. If your canary frequency is low, you must increase the frequency to increase the timeout. You can adjust both the frequency and the timeout value under **Schedule** when you create or update a canary by using the CloudWatch Synthetics console.

Be sure that your canary timeout value is no shorter than 15 seconds to allow for Lambda cold starts and the time it takes to boot up the canary instrumentation.

Canary artifacts are not available to view in the CloudWatch Synthetics console when this error happens. You can use CloudWatch Logs to see the canary's logs.

To use CloudWatch Logs to see the logs for a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **Log groups**.
3. Find the log group by typing the canary name in the filter box. Log groups for canaries have the name **/aws/lambda/cwsyn-*canaryName* -randomId**.

Trying to access an internal endpoint

If you want your canary to access an endpoint on your internal network, we recommend that you set up CloudWatch Synthetics to use VPC. For more information, see Running a canary on a VPC.

Canary runtime version upgrade and downgrade issues

If you recently upgraded the canary from runtime version `syn-1.0` to a later version, it may be a cross-origin request sharing (CORS) issue. For more information, see Cross-origin request sharing (CORS) issue.

If you recently downgraded the canary to an older runtime version, check to make sure that the CloudWatch Synthetics functions that you are using are available in the older runtime version that you downgraded to. For example, the `executeHttpStep` function is available for runtime version `syn-nodejs-2.2` and later. To check on the availability of functions, see Writing a canary script.

> **Note**
>
> When you plan to upgrade or downgrade the runtime version for a canary, we recommend that you first clone the canary and update the runtime version in the cloned canary. After you have verified that the clone with the new runtime version works, you can update the runtime version of your original canary and delete the clone.

Cross-origin request sharing (CORS) issue

In a UI canary, if some network requests are failing with `403` or `net::ERR_FAILED`, check whether the canary has active tracing enabled and also uses the Puppeteer function `page.setExtraHTTPHeaders` to add headers. If so, the failed network requests might be caused by cross-origin request sharing (CORS) restrictions. You can confirm whether this is the case by disabling active tracing or removing the extra HTTP headers.

Why does this happen?

When active tracing is used, an extra header is added to all outgoing requests to trace the call. Modifying the request headers by adding a trace header or adding extra headers using Puppeteer's `page.setExtraHTTPHeaders` causes a CORS check for XMLHttpRequest (XHR) requests.

If you don't want to disable active tracing or remove the extra headers, you can update your web application to allow cross-origin access or you can disable web security by using the `disable-web-security` flag when you launch the Chrome browser in your script.

You can override launch parameters used by CloudWatch Synthetics and pass additional `disable-web-security` flag parameters by using the CloudWatch Synthetics launch function. For more information, see Library functions available for Node.js canary scripts using Puppeteer.

> **Note**
>
> You can override launch parameters used by CloudWatch Synthetics when you use runtime version `syn-nodejs-2.1` or later.

Canary race condition issues

For the best experience when using CloudWatch Synthetics, make sure that the code written for the canaries is idempotent. Otherwise, in rare cases, canary runs may encounter race conditions when the canary interacts with the same resource across different runs.

Troubleshooting a canary on a VPC

If you have issues after creating or updating a canary on a VPC, one of the following sections might help you troubleshoot the problem.

New canary in error state or canary can't be updated

If you create a canary to run on a VPC and it immediately goes into an error state, or you can't update a canary to run on a VPC, the canary's role might not have the right permissions. To run on a VPC, a canary must have the permissions `ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces`, and `ec2:DeleteNetworkInterface`. These permissions are all contained in the `AWSLambdaVPCAccessExecutionRole` managed policy. For more information, see Execution Role and User Permissions.

If this issue happened when you created a canary, you must delete the canary, and create a new one. If you use the CloudWatch console to create the new canary, under **Access Permissions**, select **Create a new role**. A new role that includes all permissions required to run the canary is created.

If this issue happens when you update a canary, you can update the canary again and provide a new role that has the required permissions.

"No test result returned" error

If a canary displays a "no test result returned" error, one of the following issues might be the cause:

- If your VPC does not have internet access, you must use VPC endpoints to give the canary access to CloudWatch and Amazon S3. You must enable the **DNS resolution** and **DNS hostname** options in the VPC for these endpoint addresses to resolve correctly. For more information, see

Using DNS with your VPC and Using CloudWatch and CloudWatch Synthetics with interface VPC endpoints .

- Canaries must run in private subnets within a VPC. To check this, open the **Subnets** page in the VPC console. Check the subnets that you selected when configuring the canary. If they have a path to an internet gateway (**igw-**), they are not private subnets.

To help you troubleshoot these issues, see the logs for the canary.

To see the log events from a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Choose the name of the canary's log group. The log group name starts with `/aws/lambda/cwsyn-`*`canary-name`*.

Troubleshooting an auto retry canary

To understand why your canary is failing or to analyze specific failed attempts, follow these troubleshooting steps.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Choose the **Canary**.
4. Under the **Availability** tab, you can examine the run details by either:

   - Selecting a specific point on the Canary Runs graph
   - Under **Issues**, selecting a record. Note that retry attempts are tagged and share timestamps with their scheduled runs

   You can view additional information under **Steps**, **Screenshot**, **Logs**, **HAR file**, or **Traces (if active tracing is enabled)**.
5. Under **Canary artifacts and Amazon S3 location**, you can access the artifact and navigate to the Amazon S3 folders or buckets through the available links.
6. The **Canary runs** graph uses different colored points to indicate various status:

   - Blue Points – Indicates successful scheduled runs with a consistent value of 100%

- Red Points – Displays failure of both scheduled runs and all retries, marked at 0%
- Orange Points – Displays either 0% or 100%. 0% indicates ongoing retry following previous attempt failures and 100% means success was achieved after retrying

Sample code for canary scripts

This section contains code samples that illustrate some possible functions for CloudWatch Synthetics canary scripts.

Samples for Node.js and Playwright

Playwright canary with multiple steps

The following script is an example of a Node.js Playwright canary with multiple steps.

```
import { synthetics } from '@aws/synthetics-playwright';

export async function handler(event, context) {
  try {
    console.log('Running Synthetics Playwright canary');
    const browser = await synthetics.launch();
    const browserContext = await browser.newContext();
    const page = await synthetics.getPage(browserContext);


    // Add steps
    // Step 1
    await synthetics.executeStep("home-page", async () => {
      console.log("Verify home page loads")
      await page.goto('https://www.amazon.com', {waitUntil: "load"});
      await new Promise(r => setTimeout(r, 5000));
    });

    // Step 2
    await synthetics.executeStep("search", async () => {
      console.log("Searching for a product")
      const searchInput = page.getByPlaceholder("Search Amazon").first();
      await searchInput.click()
      await searchInput.fill('Amazon echo');
      const btn = page.getByRole('button', { name: 'Go' }).first()
      await btn.click({ timeout: 15000 })
      console.log("Clicked search button")
```

```
    });

    // Step 3
    await synthetics.executeStep("search-results", async () => {
      console.log("Verifying search results")
      const resultsHeading = page.getByText("Results", {exact: true}).first()
      await resultsHeading.highlight();
      await new Promise(r => setTimeout(r, 5000));
    });

  } finally {
    // Close all browser contexts and browser
    await synthetics.close();
  }
}
```

Playwright canaries setting cookies

The following script is an example of a Node.js Playwright canary setting three cookies.

```
import { synthetics } from '@aws/synthetics-playwright';

export const handler = async (event, context) => {
  try {
    let url = "http://smile.amazon.com/";
    const browser = await synthetics.launch();
    const page = await synthetics.getPage(browser);
    const cookies = [{
        'name': 'cookie1',
        'value': 'val1',
        'url': url
    },
    {
        'name': 'cookie2',
        'value': 'val2',
        'url': url
    },
    {
        'name': 'cookie3',
        'value': 'val3',
        'url': url
    }
   ];
   await page.context().addCookies(cookies);
```

```
    await page.goto(url, {waitUntil: 'load', timeout: 30000});
    await page.screenshot({ path: '/tmp/smile.png' });

  } finally {
    await synthetics.close();
  }
};
```

Samples for Node.js and Puppeteer

Setting cookies

Web sites rely on cookies to provide custom functionality or track users. By setting cookies in CloudWatch Synthetics scripts, you can mimic this custom behavior and validate it.

For example, a web site might display a **Login** link for a revisiting user instead of a **Register** link.

```
var synthetics = require('@aws/synthetics-puppeteer');
const log = require('@aws/synthetics-logger');

const pageLoadBlueprint = async function () {

    let url = "http://smile.amazon.com/";

    let page = await synthetics.getPage();

    // Set cookies.  I found that name, value, and either url or domain are required
 fields.
    const cookies = [{
      'name': 'cookie1',
      'value': 'val1',
      'url': url
    },{
      'name': 'cookie2',
      'value': 'val2',
      'url': url
    },{
      'name': 'cookie3',
      'value': 'val3',
      'url': url
    }];

    await page.setCookie(...cookies);
```

```
    // Navigate to the url
    await synthetics.executeStep('pageLoaded_home', async function (timeoutInMillis =
 30000) {

        var response = await page.goto(url, {waitUntil: ['load', 'networkidle0'],
 timeout: timeoutInMillis});

        // Log cookies for this page and this url
        const cookiesSet = await page.cookies(url);
        log.info("Cookies for url: " + url + " are set to: " +
 JSON.stringify(cookiesSet));
    });

};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Device emulation

You can write scripts that emulate various devices so that you can approximate how a page looks and behaves on those devices.

The following sample emulates an iPhone 6 device. For more information about emulation, see page.emulate(options) in the Puppeteer documentation.

```
var synthetics = require('@aws/synthetics-puppeteer');
const log = require('@aws/synthetics-logger');
const puppeteer = require('puppeteer-core');

const pageLoadBlueprint = async function () {

    const iPhone = puppeteer.devices['iPhone 6'];

    // INSERT URL here
    const URL = "https://amazon.com";

    let page = await synthetics.getPage();
    await page.emulate(iPhone);

    //You can customize the wait condition here. For instance,
```

```
    //using 'networkidle2' may be less restrictive.
    const response = await page.goto(URL, {waitUntil: 'domcontentloaded', timeout:
 30000});
    if (!response) {
        throw "Failed to load page!";
    }

    await page.waitFor(15000);

    await synthetics.takeScreenshot('loaded', 'loaded');

    //If the response status code is not a 2xx success code
    if (response.status() < 200 || response.status() > 299) {
        throw "Failed to load page!";
    }
};

exports.handler = async () => {
    return await pageLoadBlueprint();
};
```

Multi-step API canary

This sample code demonstrates an API canary with two HTTP steps: testing the same API for positive and negative test cases. The step configuration is passed to enable reporting of request/response headers. Additionally, it hides the Authorization header and X-Amz-Security-Token, because they contain user credentials.

When this script is used as a canary, you can view details about each step and the associated HTTP requests such as step pass/fail, duration, and performance metrics like DNS look up time and first byte time. You can view the number of 2xx, 4xx and 5xx for your canary run.

```
var synthetics = require('@aws/synthetics-puppeteer');
const log = require('@aws/synthetics-logger');

const apiCanaryBlueprint = async function () {

    // Handle validation for positive scenario
    const validatePositiveCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 200 || res.statusCode > 299) {
                throw res.statusCode + ' ' + res.statusMessage;
```

```
            }

            let responseBody = '';
            res.on('data', (d) => {
                responseBody += d;
            });

            res.on('end', () => {
                // Add validation on 'responseBody' here if required. For ex, your
 status code is 200 but data might be empty
                resolve();
            });
        });
    };

    // Handle validation for negative scenario
    const validateNegativeCase = async function(res) {
        return new Promise((resolve, reject) => {
            if (res.statusCode < 400) {
                throw res.statusCode + ' ' + res.statusMessage;
            }

            resolve();
        });
    };

    let requestOptionsStep1 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/product/validProductName',
        'port': 443,
        'protocol': 'https:'
    };

    let headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep1['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
```

```
    let stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };


    await synthetics.executeHttpStep('Verify GET products API with valid name',
 requestOptionsStep1, validatePositiveCase, stepConfig);

    let requestOptionsStep2 = {
        'hostname': 'myproductsEndpoint.com',
        'method': 'GET',
        'path': '/test/canary/InvalidName(',
        'port': 443,
        'protocol': 'https:'
    };

    headers = {};
    headers['User-Agent'] = [synthetics.getCanaryUserAgentString(), headers['User-
Agent']].join(' ');

    requestOptionsStep2['headers'] = headers;

    // By default headers, post data and response body are not included in the report
 for security reasons.
    // Change the configuration at global level or add as step configuration for
 individual steps
    stepConfig = {
        includeRequestHeaders: true,
        includeResponseHeaders: true,
        restrictedHeaders: ['X-Amz-Security-Token', 'Authorization'], // Restricted
 header values do not appear in report generated.
        includeRequestBody: true,
        includeResponseBody: true
    };

    await synthetics.executeHttpStep('Verify GET products API with invalid name',
 requestOptionsStep2, validateNegativeCase, stepConfig);

};
```

```
exports.handler = async () => {
    return await apiCanaryBlueprint();
};
```

Samples for Python and Selenium

The following sample Selenium code is a canary that fails with a custom error message when a target element is not loaded.

```
from aws_synthetics.selenium import synthetics_webdriver as webdriver
from aws_synthetics.common import synthetics_logger as logger
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

def custom_selenium_script():
    # create a browser instance
    browser = webdriver.Chrome()
    browser.get('https://www.example.com/')
    logger.info('navigated to home page')
    # set cookie
    browser.add_cookie({'name': 'foo', 'value': 'bar'})
    browser.get('https://www.example.com/')
    # save screenshot
    browser.save_screenshot('signed.png')
    # expected status of an element
    button_condition = EC.element_to_be_clickable((By.CSS_SELECTOR, '.submit-button'))
    # add custom error message on failure
    WebDriverWait(browser, 5).until(button_condition, message='Submit button failed to
 load').click()
    logger.info('Submit button loaded successfully')
    # browser will be quit automatically at the end of canary run,
    # quit action is not necessary in the canary script
    browser.quit()

# entry point for the canary
def handler(event, context):
    return custom_selenium_script()
```

Canaries and X-Ray tracing

You can choose to enable active AWS X-Ray tracing on canaries that use the `syn-nodejs-2.0` or later runtime. With tracing enabled, traces are sent for all calls made by the canary that use the browser, the AWS SDK, or HTTP or HTTPS modules. Canaries with tracing enabled appear on the X-Ray Trace Map, and within Application Signals after you have enabled it for your application.

> **Note**
>
> Activating X-Ray tracing on canaries is not yet supported in Asia Pacific (Jakarta).
> X-Ray tracing is not supported with Firefox browser canaries.

When a canary appears on the X-Ray trace map, it appears as a new client node type. You can hover on a canary node to see data about latency, requests, and faults. You can also choose the canary node to see more data at the bottom of the page. From this area of the page, you can choose **View in Synthetics** to jump to the CloudWatch Synthetics console for more details about the canary, or choose **View Traces** to see more details about the traces from this canary's runs.

A canary with tracing enabled also has a **Tracing** tab in its details page, with details about traces and segments from the canary's runs.

Enabling tracing increases canary run time by 2.5% to 7%.

A canary with tracing enabled must use a role with the following permissions. If you use the console to create the role when you create the canary, it is given these permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Sid230934",
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Traces generated by canaries incur charges. For more information about X-Ray pricing, see AWS X-Ray Pricing.

Running a canary on a VPC

You can run canaries on endpoints on a VPC and public internal endpoints. To run a canary on a VPC, you must have both the **DNS Resolution** and **DNS hostnames** options enabled on the VPC. For more information, see Using DNS with your VPC.

When you run a canary on a VPC endpoint, you must provide a way for it to send its metrics to CloudWatch and its artifacts to Amazon S3. If the VPC is already enabled for internet access, there's nothing more for you to do. The canary executes in your VPC, but can access the internet to upload its metrics and artifacts.

If the VPC is not already enabled for internet access, you have two options:

- Enable IPv4 internet access to allow the canary to send metrics to CloudWatch and Amazon S3. For more information, see the following section Giving internet access to your canary on a VPC.
- If you want to keep your VPC private, you can configure the canary to send its data to CloudWatch and Amazon S3 through private VPC endpoints. If you have not already done so, you must create a VPC endpoint for CloudWatch (com.amazonaws.*region*.monitoring) and a gateway endpoint for Amazon S3. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints and Amazon VPC Endpoints for Amazon S3.

Giving internet access to your canary on a VPC

Follow these steps to give internet access to your VPC canary, or to assign your canary a static IP address

To give internet access (IPv4) to a canary on a VPC

1. Create a NAT gateway in a public subnet on the VPC. For instructions, see Create a NAT gateway.

2. Add a new route to the route table in the private subnet where the canary is launched. Specify the following:

   - For **Destination**, enter **`0.0.0.0/0`**
   - For **Target**, choose **NAT Gateway**, and then choose the ID of the NAT gateway that you created.
   - Choose **Save routes**.

   For more information about adding the route to the route table, see Add and remove routes from a route table.

To give internet access (IPv6) to a canary on a VPC

1. Configure your VPC to have Dualstack subnets. You must add an Egress-Only Internet Gateway to the VPC, update the route tables to allow traffic to the Internet Gateway, and allow outbound access from the associated Security Groups. For more information, see Add IPv6 support for your VPC .
2. Set the `Ipv6AllowedForDualstack` in your canary VPC configuration using the `CreateCanary` or `UpdateCanary` API. For more information, see VpcConfigInput .

   To enable outbound IPv6 traffic from your canary, the VPC subnets attached to the canary must be enabled for Dualstack.

> **Note**
>
> Be sure that the routes to your NAT gateway are in an **active** status. If the NAT gateway is deleted and you haven't updated the routes, they're in a black hole status. For more information, see Work with NAT gateways.

Encrypting canary artifacts

CloudWatch Synthetics stores canary artifacts such as screenshots, HAR files, and reports in your Amazon S3 bucket. By default, these artifacts are encrypted at rest using an AWS managed key. For more information, see Customer keys and AWS keys .

You can choose to use a different encryption option. CloudWatch Synthetics supports the following:

- **SSE-S3**– Server-side encryption (SSE) with an Amazon S3-managed key.
- **SSE-KMS**– Server-side encryption (SSE) with an AWS KMS customer managed key.

If you want to use the default encryption option with an AWS managed key, you don't need any additional permissions.

To use SSE-S3 encryption, you specify **SSE_S3** as the encryption mode when you create or update your canary. You do not need any additional permissions to use this encryption mode. For more information, see Protecting data using server-side encryption with Amazon S3-managed encryption keys (SSE-S3).

To use an AWS KMS customer managed key, you specify **SSE-KMS** as the encryption mode when you create or update your canary, and you also provide the Amazon Resource Name (ARN) of your key. You can also use a cross-account KMS key.

To use a customer managed key, you need the following settings:

- The IAM role for your canary must have permission to encrypt your artifacts using your key. If you are using visual monitoring, you must also give it permission to decrypt artifacts.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowKMSKeyUsage",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

- Instead of adding permissions to your IAM role, you can add your IAM role to your key policy. If you use the same role for multiple canaries, you should consider this approach.

```
{
    "Sid": "Enable IAM User Permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "Your synthetics IAM role ARN"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

- If you are using a cross-account KMS key, see Allowing users in other accounts to use a KMS key.

Viewing encrypted canary artifacts when using a customer managed key

To view canary artifacts, update your customer managed key to give AWS KMS the decrypt permission to the user viewing the artifacts. Alternatively, add decrypt permissions to the user or IAM role that is viewing the artifacts.

The default AWS KMS policy enables IAM policies in the account to allow access to the KMS keys. If you are using a cross-account KMS key, see Why are cross-account users getting Access Denied errors when they try to access Amazon S3 objects encrypted by a custom AWS KMS key?.

For more information about troubleshooting access denied issues because of a KMS key, see Troubleshooting key access.

Updating artifact location and encryption when using visual monitoring

To perform visual monitoring, CloudWatch Synthetics compares your screenshots with baseline screenshots acquired in the run selected as the baseline. If you update your artifact location or encryption option, you must do one of the following:

- Make sure that your IAM role has sufficient permission for both the previous Amazon S3 location and the new Amazon S3 location for artifacts. Also make sure that it has permission for both the previous and new encryption methods and KMS keys.
- Create a new baseline by selecting the next canary run as a new baseline. If you use this option, you only need to make sure that your IAM role has sufficient permissions for the new artifact location and encryption option.

We recommend the second option of selecting the next run as the new baseline. This avoids having a dependency on an artifact location or encryption option that you're not using anymore for the canary.

For example, suppose that your canary uses artifact location A and KMS key K for uploading artifacts. If you update your canary to artifact location B and KMS key L, you can make sure that your IAM role has permissions to both of the artifact locations (A and B) and both of the KMS keys (K and L). Alternatively, you can select the next run as the new baseline and make sure that your canary IAM role has permissions to artifact location B and KMS key L.

Viewing canary statistics and details

You can view details about your canaries and see statistics about their runs.

To be able to see all the details about your canary run results, you must be logged on to an account that has sufficient permissions. For more information, see Required roles and permissions for CloudWatch canaries.

To view canary statistics and details

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.

   In the details about the canaries that you have created:

   - **Status** visually shows how many of your canaries have passed their most recent runs.
   - **Groups** displays the groups you have created, and displays how many of them have failing or alarming canaries.
   - **Slowest performers** displays the group and the Region with the slowest-performing canaries. These are calculated by adding up the average duration of all canaries (across the time span selected) within a group or Region and dividing it by the number of canaries in the

group or Region. If you choose the metric for Slowest group, the table is filtered to display only the slowest groups and their canaries. The table is sorted by Average Duration.

- Near the bottom of the page is a table displaying all canaries. You can use the filtering bar to filter the table to show canaries by specific canary names, last run results, success percentage, alarms, run rates, canary state, runtimes, and unique tags.

  For the alarms column, only alarms that conform to the naming standard for canary alarms are displayed. This standard is `Synthetics-Alarm-`*`canaryName`*` -`*`index`*. Canary alarms that you create in the **Synthetics** section of the CloudWatch console automatically use this naming convention. If you create canary alarms in the **Alarms** section of the CloudWatch console or by using AWS CloudFormation, and you don't use this naming convention, the alarms work but they do not appear in this list.

3. To see more details about a single canary, choose the name of the canary in the **Canaries** table.

   In the details about that canary:

   - The **Availability** tab displays information about the recent runs of this canary.

     Under **Canary runs**, you can choose one of the lines to see details about that run.

     Under the graph, you can choose **Steps**, **Screenshot** , **Logs**, or **HAR file** to see these types of details. If the canary has active tracing enabled, you can also choose **Traces** to see tracing information from the canary's runs.

     The logs for canary runs are stored in S3 buckets and in CloudWatch Logs.

     Screenshots show how your customers view your webpages. You can use the HAR files (HTTP Archive files) to view detailed performance data about the webpages. You can analyze the list of web requests and catch performance issues such as time to load for an item. Log files show the record of interactions between the canary run and the webpage and can be used to identify details of errors.

     If the canary uses the `syn-nodejs-2.0-beta` runtime or later, you can sort the HAR files by status code, request size, or duration.

     The **Steps** tab displays a list of the canary's steps, each step's status, failure reason, URL after step execution, screenshots, and duration of step execution. For API canaries with HTTP

steps, you can view steps and corresponding HTTP requests if you are using runtime `syn-nodejs-2.2` or later.

Choose the **HTTP Requests** tab to view the log of each HTTP request made by the canary. You can view request/response headers, response body, status code, error and performance timings (total duration, TCP connection time, TLS handshake time, first byte time, and content transfer time). All HTTP requests which use the HTTP/HTTPS module under the hood are captured here.

By default in API canaries, the request header, response header, request body, and response body are not included in the report for security reasons. If you choose to include them, the data is stored only in your S3 bucket. For information about how to include this data in the report, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]).

Response body content types of text, HTML and JSON are supported. Content types like text/HTML, text/plain, application/JSON and application/x-amz-json-1.0 are supported. Compressed responses are not supported.

- The **Monitoring** tab displays graphs of the CloudWatch metrics published by this canary. For more information about these metrics, see CloudWatch metrics published by canaries.

  Below the CloudWatch graphics published by the canary are graphs of Lambda metrics related to the canary's Lambda code.
- The **Configuration** tab displays configuration and schedule information about the canary.
- The **Groups** tab displays the groups that this canary is associated with, if any.
- The **Tags** tab displays the tags associated with the canary.

CloudWatch metrics published by canaries

Canaries publish the following metrics to CloudWatch in the `CloudWatchSynthetics` namespace. For more information about viewing CloudWatch metrics, see View available metrics .

> **Note**
>
> For multi-browser canaries, browser-dimension metrics are enabled by default to provide visibility into performance across browsers like Chrome, Firefox and other browsers. To disable browser metrics, set `browserDimension` to `false`.

For single-browser canaries, browser-dimension metrics are disabled by default to avoid redundancy. To see metrics broken down by browser, set `browserDimension` to `true`.

Metric	Description
2xx	The number of network requests performed by the canary that returned OK responses, with response codes between 200 and 299. This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName, Browser Valid Statistic: Sum Units: Count
4xx	The number of network requests performed by the canary that returned Error responses, with response codes between 400 and 499. This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName, Browser Valid Statistic: Sum Units: Count
5xx	The number of network requests performed by the canary that returned Fault responses, with response codes between 500 and 599. This metric is reported for UI canaries that use runtime version `syn-nodejs-2.0` or later, and is reported for API canaries that use runtime version `syn-nodejs-2.2` or later. Valid Dimensions: CanaryName, Browser

Metric	Description
	Valid Statistic: Sum Units: Count
`Duration`	The duration in milliseconds of the canary run. Valid Dimensions: CanaryName, Browser, Location Valid Statistic: Average Units: Milliseconds
`DurationDryRun`	The duration of DryRun executions. Valid Dimensions: CanaryName, Browser Valid Statistic: Average Units: Milliseconds
`EphemeralStorageUsagePercent`	The maximum percentage of ephemeral storage used compared to total ephemeral storage configured. This metric is collected at every 10 second interval.
`Failed`	The number of canary runs that failed to execute. These failures are related to the canary itself. Valid Dimensions: CanaryName, Browser Valid Statistic: Sum Units: Count

Metric	Description
`Failed requests`	The number of HTTP requests executed by the canary on the target website that failed with no response. Valid Dimensions: CanaryName, Browser Valid Statistic: Sum Units: Count
`RetryCount`	The number of times your canary retried. This metric is only displayed when there are retries. Valid Dimensions: CanaryName, Browser Valid Statistic: Sum Units: Count
`SuccessPercent`	The percentage of the runs of this canary that succeed and find no failures. Valid Dimensions: CanaryName, Browser, Location Valid Statistic: Average Units: Percent
`SuccessPercentDryRun`	The success percentage of DryRun executions. Valid Dimensions: CanaryName, Browser Valid Statistic: Average Units: Percent

Metric	Description
SuccessPercentWithRetries	The percentage of the runs of this canary that succeed after all attempts. Valid Dimensions: CanaryName, Browser, Location Valid Statistic: Average Units: Percent
VisualMonitoringSuccessPercent	The percentage of visual comparisons that successfully matched the baseline screenshots during a canary run. Valid Dimensions: CanaryName, Browser Valid Statistic: Average Units: Percent
VisualMonitoringTotalComparisons	The total number of visual comparisons that happened during a canary run. Valid Dimensions: CanaryName, Browser Units: Count

Note

Canaries that use either the `executeStep()` or `executeHttpStep()` methods from the Synthetics library also publish `SuccessPercent` and `Duration` metrics with the dimensions `CanaryName` and `StepName` for each step.

Edit or delete a canary

You can edit or delete an existing canary.

Edit canary

When you edit a canary, even if you don't change its schedule, the schedule is reset corresponding to when you edit the canary. For example, if you have a canary that runs every hour, and you edit that canary, the canary will run immediately after the edit is completed and then every hour after that.

To edit or update a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Select the button next to the canary name, and choose **Actions**, **Edit**.
4. (Optional) If this canary performs visual monitoring of screenshots and you want to set the next run of the canary as the baseline, select **Set next run as new baseline**.
5. (Optional) If this canary performs visual monitoring of screenshots and you want to remove a screenshot from visual monitoring or you want to designate parts of the screenshot to be ignored during visual comparisons, under **Visual Monitoring** choose **Edit Baseline**.

   The screenshot appears, and you can do one of the following:

   - To remove the screenshot from being used for visual monitoring, select **Remove screenshot from visual test baseline**.
   - To designate parts of the screenshot to be ignored during visual comparisons, click and drag to draw areas of the screen to ignore. After you have done this for all the areas that you want to ignore during comparisons, choose **Save** .
6. Under **Script editor**, **Runtime version**, select a synthetics runtime version to execute the canary. For information on synthetics runtime versions, see Synthetics runtime versions .

   Under **Browser configuration**, you can enable the browser to test the canary. You must select at least one browser.
7. (Optional) To add or remove replica locations from a multilocation canary, update the selections under **Locations**. You can only edit replica locations from the primary Region. For more information, see Adding or removing replica locations.
8. Make any other changes to the canary that you'd like, and choose **Save** .

Delete canary

When you delete a canary, you can choose whether to also delete other resources used and created by the canary. If the canary's `ProvisionedResourceCleanup` field is set to `AUTOMATIC` or

`DeleteLambda` is specified as `true` when you delete the canary, CloudWatch Synthetics will automatically delete the Lambda functions and layers that are used by the canary.

When you delete a canary, you should also delete the following:

- Lambda functions and layers used by this canary. Their prefix is `cwsyn-` *`MyCanaryName`*.
- CloudWatch alarms created for this canary. These alarms have a name that starts with `Synthetics-Alarm-`*`MyCanaryName`*. For more information about deleting alarms, see Edit or delete a CloudWatch alarm.
- Amazon S3 objects and buckets, such as the canary's results location and artifact location.
- IAM roles created for the canary. These have the name `role/service-role/CloudWatchSyntheticsRole-`*`MyCanaryName`*.
- Log groups in CloudWatch Logs created for the canary. These logs groups have the following names: `/aws/lambda/cwsyn-`*`MyCanaryName-randomId`* .

Before you delete a canary, you might want to view the canary details and make note of this information. That way, you can delete the correct resources after you delete the canary.

To delete a canary

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. If the canary is currently in the `RUNNING` state, you must stop it. Only canaries in the `STOPPED`, `READY(NOT_STARTED)`, or `ERROR` states can be deleted.

   To stop the canary, select the button next to the canary name, and choose **Actions**, **Stop**.
4. (Optional) If this is a multilocation canary, you must first remove all replicas. Choose **Actions**, **Edit**, and clear all replica Regions under **Locations**. Save your changes and wait for all replicas to be removed. For more information, see Deleting a multilocation canary.
5. Select the button next to the canary name, and choose **Actions**, **Delete**.
6. Choose whether to also delete the other resources created for and used by the canary. Lambda functions and layers will be deleted alongside the canary, but you can additionally choose to delete the canary's IAM role and IAM policy.

   Enter **`Delete`** into the box and choose **Delete** .
7. Delete the other resources used by and created for the canary, as listed earlier in this section.

Start, stop, delete, or update runtime for multiple canaries

You can stop, start, delete, or update the runtime of as many as five canaries with one action. If you update the runtime of a canary, it is updated to the latest runtime available for the language and framework that the canary uses.

If you select multiple canaries and only some of them are in a state that is valid for the action that you select, the action is performed only on the canaries where that action is valid. For example, if you select some canaries that are currently running and some that are not, and you select to start the canaries, the canaries that weren't already running will start, and the canaries that were already running are not affected.

If none of the canaries that you select are valid for an action, that action will not be available in the menu.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **Synthetics Canaries**.
3. Select the check boxes next to the canaries that you want to stop, start, or delete.
4. Choose **Actions** and then choose either **Start**, **Stop**, **Delete**, **Start Dry Run**, or **Update Runtime**.

   In addition, when choosing **Update Runtime**, you can choose to dry run the runtime update first before committing the change.

Monitoring canary events with Amazon EventBridge

Amazon EventBridge event rules can notify you when canaries change status or complete runs. EventBridge delivers a near-real-time stream of system events that describe changes in AWS resources. CloudWatch Synthetics sends these events to EventBridge on a *best effort* basis. Best effort delivery means that CloudWatch Synthetics attempts to send all events to EventBridge, but in some rare cases an event might not be delivered. EventBridge processes all received events at least once. Additionally, your event listeners might not receive the events in the order that the events occurred.

> **Note**
>
> Amazon EventBridge is an event bus service that you can use to connect your applications with data from a variety of sources. For more information, see What is Amazon EventBridge? in the *Amazon EventBridge User Guide*.

CloudWatch Synthetics emits an event when a canary changes state or completes a run. You can create an EventBridge rule that includes an event pattern to match all event types sent from CloudWatch Synthetics, or that matches only specific event types. When a canary triggers a rule, EventBridge invokes the target actions defined in the rule. This allows you to send notifications, capture event information, and take corrective action, in response to a canary state change or the completion of a canary run. For example, you can create rules for the following use cases:

- Investigating when a canary run fails
- Investigating when a canary has gone into the ERROR state
- Tracking a canary's life cycle
- Monitoring canary run success or failure as part of a workflow

Example events from CloudWatch Synthetics

This section lists example events from CloudWatch Synthetics. For more information about event format, see Events and Event Patterns in EventBridge.

Canary status change

In this event type, the values of `current-state` and `previous-state` can be the following:

`CREATING` | `READY` | `STARTING` | `RUNNING` | `UPDATING` | `STOPPING` | `STOPPED` | `ERROR`

```
{
                "version": "0",
                "id": "8a99ca10-1e97-2302-2d64-316c5dedfd61",
                "detail-type": "Synthetics Canary Status Change",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:19:43Z",
                "region": "us-east-1",
                "resources": [],
```

```
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "current-state": "STOPPED",
                                "previous-state": "UPDATING",
                                "source-location": "NULL",
                                "updated-on": 1612909161.767,
                                "changed-config": {
                                                "executionArn": {
                                                                "previous-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:function:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                },
                                                "vpcId": {
                                                                "current-value": "NULL"
                                                },
                                                "testCodeLayerVersionArn": {
                                                                "previous-
value": "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:1",
                                                                "current-value":
 "arn:aws:lambda:us-east-1:123456789012:layer:cwsyn-events-bb-1-af3e3a05-
dc5a-4f5f-96d1-989EXAMPLE:2"
                                                }
                                },
                                "message": "Canary status has changed"
                }
}
```

Successful canary run completed

```
{
                "version": "0",
                "id": "989EXAMPLE-f4a5-57a7-1a8f-d9cc768a1375",
                "detail-type": "Synthetics Canary TestRun Successful",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:01Z",
                "region": "us-east-1",
```

```
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "989EXAMPLE-dc5a-4f5f-96d1-989b75a94226",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "c6c39152-8f4a-471c-9810-989EXAMPLE",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-28ddbe266797/2021/02/09/22/23-41-200",
                                "test-run-status": "PASSED",
                                "state-reason": "null",
                                "canary-run-timeline": {
                                                "started": 1612909421,
                                                "completed": 1612909441
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

Failed canary run completed

```
{
                "version": "0",
                "id": "2644b18f-3e67-5ebf-cdfd-bf9f91392f41",
                "detail-type": "Synthetics Canary TestRun Failure",
                "source": "aws.synthetics",
                "account": "123456789012",
                "time": "2021-02-09T22:24:27Z",
                "region": "us-east-1",
                "resources": [],
                "detail": {
                                "account-id": "123456789012",
                                "canary-id": "af3e3a05-dc5a-4f5f-96d1-9989EXAMPLE",
                                "canary-name": "events-bb-1",
                                "canary-run-id": "0df3823e-7e33-4da1-8194-
b04e4d4a2bf6",
                                "artifact-location": "cw-syn-results-123456789012-us-
east-1/canary/us-east-1/events-bb-1-ec3-989EXAMPLE/2021/02/09/22/24-21-275",
                                "test-run-status": "FAILED",
                                "state-reason": "\"Error: net::ERR_NAME_NOT_RESOLVED
 \""
                                "canary-run-timeline": {
                                                "started": 1612909461,
                                                "completed": 1612909467
```

```
                                },
                                "message": "Test run result is generated successfully"
                }
}
```

It's possible that events might be duplicated or out of order. To determine the order of events, use the `time` property.

Prerequisites for creating EventBridge rules

Before you create an EventBridge rule for CloudWatch Synthetics, you should do the following:

- Familiarize yourself with events, rules, and targets in EventBridge.
- Create and configure the targets invoked by your EventBridge rules. Rules can invoke many types of targets, including:
  - Amazon SNS topics
  - AWS Lambda functions
  - Kinesis streams
  - Amazon SQS queues

For more information, see What is Amazon EventBridge? and Getting started with Amazon EventBridge in the *Amazon EventBridge User Guide*.

Create an EventBridge rule (CLI)

The steps in the following example create an EventBridge rule that publishes an Amazon SNS topic when the canary named `my-canary-name` in `us-east-1` completes a run or changes state.

1. Create the rule.

   ```
   aws events put-rule \
     --name TestRule \
     --region us-east-1 \
     --event-pattern "{\"source\": [\"aws.synthetics\"], \"detail\": {\"canary-name\":
    [\"my-canary-name\"]}}"
   ```

   Any properties you omit from the pattern are ignored.

2. Add the topic as a rule target.

- Replace *topic-arn* with the Amazon Resource Name (ARN) of your Amazon SNS topic.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> To allow Amazon EventBridge to call your target topic, you must add a resource-based policy to your topic. For more information, see Amazon SNS permissions in the *Amazon EventBridge User Guide*.

For more information, see Events and event patterns in EventBridge in the *Amazon EventBridge User Guide*.

Performing safe canary updates

CloudWatch synthetics safe canary updates allows you to test the updates on your existing canaries before applying the changes. This feature helps you validate canary compatibility with new run times and other configuration changes such as code or memory changes. This will help minimize potential monitoring disruptions caused by erroneous updates.

By using canary safe updates on runtime version updates, configuration changes, and code script modifications, you can mitigate risk, maintain uninterrupted monitoring, verify the changes before committing, update, and reduce downtime.

Topics

- Prerequisites
- Best practices
- Testing canary using dry run
- Limitations

Prerequisites

Make sure the prequisites are complete.

- AWS account with CloudWatch synthetics permissions
- Existing canary on the supported runtime versions (see Limitations for compatible runtimes)
- Include compatible runtimes when performing a dry run (see Limitations for compatible runtimes)

Best practices

Here are some best practices to follow while performing a canary .

- Execute a dry run to validate a runtime update
- Perform dry runs before production updates to canary
- Review canary logs and artifacts after a dry run
- Use dry runs to validate dependencies and library compatibility

Testing canary using dry run

You can test the canary update using the following options:

Using the AWS Management Console's Edit workflow

1. Go the CloudWatch synthetics console.
2. Select the canary you want to update.
3. From the **Actions** drop down, choose **Edit**.

   Update the canary with the changes you want to test. For example, changing runtime version or editing the script's code.
4. Under **Canary script**, choose **Start Dry Run** to test and view the results immediately or choose **Validate and save later** at the bottom of the page to start the test and view the results later in your **Canary Details** page.
5. After the dry run succeeds, choose **Submit** to commit your canary updates.

Using the AWS Management Console for updating canaries in a batch

1. Go the CloudWatch synthetics console.
2. Choose the **Synthetics** list page.

3. Select upto five canaries for which you want to update the runtime.
4. From the **Actions** drop down, choose **Update Runtime** .
5. Choose **Start dry run for new runtime** to start the dry run and test your changes before an update.
6. On the **Synthetics** list page, you will see a text next to the **Runtime** version for the canary that displays the progress of the dry run (this is only displayed for dry runs involving a runtime update).

   After the dry run succeeds, you will see an **Initiate Update** text.
7. Choose **Initiate Update** to commit the runtime update.
8. If the dry run fails, you will see an **Update dry run failed** text. Choose the text to view the debug link to the canary details page.

Using the AWS CLI or SDK

The API starts the dry run for the provided canary name `MyCanary` and updates the runtime version to `syn-nodejs-puppeteer-10.0`.

```
aws synthetics start-canary-dry-run \
    --name MyCanary \
    --runtime-version syn-nodejs-puppeteer-10.0

      // Or if you wanted to update other configurations:

aws synthetics start-canary-dry-run \
    --name MyCanary \
    --execution-role-arn arn:aws:iam::123456789012:role/NewRole
```

The API will return the `DryRunId` inside the `DryRunConfigOutput` .

Call `GetCanary` with the provided `DryRunId` to receive the canary's dry run configurations and an additional field `DryRunConfig` which contains the status of the dry run listed as `LastDryRunExecutionStatus`.

```
aws synthetics get-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

For more details, use `GetCanaryRuns` with the provided `DryRunId` to retrieve the run and additional information.

```
aws synthetics get-canary-runs \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

After a successful dry run, you can then use `UpdateCanary` with the provided `DryRunId` in order to commit your changes.

```
aws synthetics update-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

When it fails for any reason (result from GetCanaryRuns will have the details), the result from `GetCanaryRuns` has an artifact location that contains logs to debug. When there are no logs, the dry run failed to be created. You can validate by using `GetCanary`.

```
aws synthetics get-canary \
    --name MyCanary \
    --dry-run-id XXXX-XXXX-XXXX-XXXX
```

The *State*, *StateReason*, and *StateReasonCode* displays the status of the dry run.

Using CloudFormation

In your template for a Synthetics Canary, provide the field `DryRunAndUpdate` which accepts a boolean value `true` or `false`.

when the value is `true` every update executes a dry run to validate the changes before automatically updating the canary. When the dry run fails, the canary does not update and fails the deployment and CloudFormation deployment with a valid reason. To debug this issue, use the AWS Synthetics console or if using an API, get the `ArtifactS3Location` using the `GetCanaryRuns` API, and download the `*-log.txt` files to review the canary log executions for errors. After validation, modify the CloudFormation template and retry the deployment or use the above API to validate.

When the value is `false`, synthetics will not execute a dry run to validate changes and will directly commit your updates.

For information on troubleshooting a failed canary, see Troubleshooting a failed canary.

An example template.

```
SyntheticsCanary:
    Type: 'AWS::Synthetics::Canary'
    Properties:
      Name: MyCanary
      RuntimeVersion: syn-nodejs-puppeteer-10.0
      Schedule: {Expression: 'rate(5 minutes)', DurationInSeconds: 3600}
      ...
      DryRunAndUpdate: true
```

Limitations

- Supports runtime versions – syn-nodejs-puppeteer-10.0+, syn-nodejs-playwright-2.0+, syn-python-selenium-5.1+, and syn-nodejs-3.0+
- You can only execute one dry run per canary at a time
- When a dry run fails, you cannot update the canary
- Dry run cannot test any **Schedule** field changes

> **Note**
>
> When you initiate a dry run with code changes for a Playwright canary and you want to update the canary without providing the associated `DryRunId`, you must explicitly specify the code parameters.

CloudWatch RUM

With CloudWatch RUM, you can perform real user monitoring to collect and view client-side data about your web and mobile application performance from actual user sessions in near real time. For Web applications, you can analyze page load times, client-side errors, and user behavior. For Mobile applications, you can monitor screen load times, app launch times, network errors, crashes, and platform-specific issues such as Android Application Not Responding (ANR) and iOS App Hangs. When you view this data, you can see it all aggregated together and also see breakdowns by device types, operating systems, and other characteristics of your application usage.

You can use the collected data to quickly identify and debug client-side performance issues. CloudWatch RUM helps you visualize anomalies in your application performance and find relevant debugging data such as error messages, stack traces, and user sessions. You can also use RUM to understand the range of end user impact including the number of users, geolocations, and browsers/devices used.

End user data that you collect for CloudWatch RUM is retained for 30 days and then automatically deleted. If you want to keep the RUM telemetry data for a longer time, you can choose to have the app monitor send copies of the telemetry to CloudWatch Logs in your account. Then, you can adjust the retention period for that log group.

To use RUM, you create an *app monitor* and provide some information. RUM generates a code snippet that you can use to add a dependency injection into your application. The snippet pulls in the RUM client code as needed. The RUM client captures data from a percentage of your application's user sessions, which is displayed in a pre-built dashboard. You can specify what percentage of user sessions to gather data from.

CloudWatch RUM is integrated with Application Signals, which can discover and monitor your application services, clients, Synthetics canaries, and service dependencies. Use Application Signals to see a list or visual map of your services, view health metrics based on your service level objectives (SLOs), and drill down to see correlated X-Ray traces for more detailed troubleshooting. To see RUM client page requests in Application Signals, turn on X-Ray active tracing when creating an app monitor. For web applications, you can also enable this by manually configuring the RUM web client. Your RUM clients are displayed on the Application Map connected to your services, and in the Service detail page of the services they call.

The RUM clients are open source. For more information, see CloudWatch RUM web client, the AWS Distro for OpenTelemetry (ADOT) Android SDK, and AWS Distro for OpenTelemetry (ADOT) iOS SDK.

RUM Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Region availability

CloudWatch RUM is currently available in the following Regions:

- US East (N. Virginia)

- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- Asia Pacific (Mumbai)
- Asia Pacific (Hyderabad)
- Asia Pacific (Melbourne)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Jakarta)
- Asia Pacific (Malaysia)
- Asia Pacific (Thailand)
- Asia Pacific (Tokyo)
- Asia Pacific (Hong Kong)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- AWS European Sovereign Cloud (Germany)
- Middle East (Bahrain)

- Middle East (UAE)
- Mexico (Central)
- South America (São Paulo)
- Israel (Tel Aviv)
- Canada West (Calgary)

Set up a mobile application to use CloudWatch RUM

To monitor mobile applications, you create an app monitor, configure it for mobile platforms, and integrate the AWS Distro for OpenTelemetry (ADOT) SDK into your application. Mobile RUM uses the OpenTelemetry Protocol (OTLP) to send telemetry data to a dedicated OTLP endpoint.

To create an app monitor for a mobile platform

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **Add app monitor**.
4. For **App monitor name**, enter a name to be used to identify this app monitor within the CloudWatch RUM console.
5. Select **Android** or **iOS** as the platform.
6. Under **Data storage**, you can choose to store copies of RUM OTEL log events and spans in CloudWatch Logs and configure retention. By default, the CloudWatch Logs log group retains the data for 30 days. You can adjust the retention period in the CloudWatch Logs console.
7. (Optional) Under **Resource Based Policy**, choose to add a resource-based policy to control who can send requests to your app monitor. If you choose **Create public policy**, a resource policy will be attached that enables anyone to send requests to your app monitor. For more information, see Using resource-based policies with CloudWatch RUM.
8. To enable AWS X-Ray tracing of sampled user sessions, choose **Active tracing** and select **Trace my service with AWS X-Ray**.

   If selected, OTEL spans generated during sampled user sessions are traced. You can then see traces and spans from these sessions in the RUM dashboard, and the X-Ray trace map and trace details pages. These user sessions will also show up as client pages in Application Signals after you have enabled it for your application.

9. (Optional) To add tags to the app monitor:

   a. Choose **Tags**, **Add new tag**.

   b. For **Key**, enter a name for the tag. You can add an optional value in **Value**.

   c. To add another tag, choose **Add new tag** again.

   For more information, see Tagging AWS Resources in the *AWS Tagging and Tag Editor User Guide*.

10. Choose **Add app monitor**.

11. In the **Sample code** section, you can copy the code snippet to add to your application. Using the AWS Distro for OpenTelemetry (ADOT) SDK, you can choose between **Manual Instrumentation** to configure the monitoring in your application code, or **Zero-Code Instrumentation** which requires minimal configuration changes.

    For both Android and iOS applications, Zero-Code Instrumentation is the simplest option as it automatically initializes telemetry collection using a configuration file. Manual Instrumentation gives you more control over the initialization and configuration process.

12. Choose **Copy** or **Download**, and then choose **Done**.

iOS Application Setup

For iOS applications, integrate the AWS Distro for OpenTelemetry (ADOT) iOS SDK to enable RUM monitoring. The SDK supports iOS 16 and later versions and provides automatic instrumentation for common performance scenarios.

Android Application Setup

For Android applications, integrate the AWS Distro for OpenTelemetry (ADOT) Android SDK to enable RUM monitoring. The SDK provides automatic instrumentation and supports both signed and unsigned authentication models.

Authentication and Security

Mobile RUM supports flexible authentication models as defined in their SDKs.

- iOS applications use the AWS Distro for OpenTelemetry (ADOT) iOS SDK.
- Android applications use the AWS Distro for OpenTelemetry (ADOT) Android SDK.

IAM policies to use CloudWatch RUM

To be able to fully manage CloudWatch RUM, you must be signed in as an IAM user or role that has the **AmazonCloudWatchRUMFullAccess** IAM policy. Additionally, you may need other policies or permissions:

- To create an app monitor that creates a new Amazon Cognito identity pool for authorization, you need to have the **Admin** IAM role or the **AdministratorAccess** IAM policy.
- To create an app monitor that sends data to CloudWatch Logs, you must be logged on to an IAM role or policy that has the following permission:

```
{
    "Effect": "Allow",
    "Action": [
        "logs:PutResourcePolicy"
    ],
    "Resource": [
        "*"
    ]
}
```

- To enable JavaScript source maps in an app monitor, you will need to upload your source map files to a Amazon S3 bucket. Your IAM role or policy needs specific Amazon S3 permissions that allow creating Amazon S3 buckets, setting bucket policies, and managing files in the bucket. For security, scope these permissions to specific resources. The example policy below restricts access to buckets containing `rum` in their names and uses the `aws:ResourceAccount` condition key to limit permissions to the principal account only.

```
{
    "Sid": "AllowS3BucketCreationAndListing",
    "Effect": "Allow",
    "Action": [
        "s3:CreateBucket",
        "s3:ListAllMyBuckets"
    ],
    "Resource": "arn:aws:s3:::*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
```

```
},
{
    "Sid": "AllowS3BucketActions",
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation",
        "s3:ListBucket"
    ],
    "Resource": "arn:aws:s3:::*rum*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "AllowS3BucketPolicyActions",
    "Effect": "Allow",
    "Action": [
        "s3:PutBucketPolicy",
        "s3:GetBucketPolicy"
    ],
    "Resource": "arn:aws:s3:::*rum*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
},
{
    "Sid": "AllowS3ObjectActions",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload"
    ],
    "Resource": "arn:aws:s3:::*rum*",
    "Condition": {
        "StringEquals": {
            "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
    }
```

```
}
```

- To use your own AWS KMS keys for server-side encryption on your source map bucket, your IAM role or policy will need specific AWS KMS permissions that allows creating a key, updating the key policy, using the AWS KMS key with Amazon S3 and setting the encryption configuration of your Amazon S3 bucket. For security, scope these permissions to specific purposes. The example below restricts access to keys for a specific region and accountId and has similar S3 restrictions as the above example.

```
{
    "Sid": "AllowKMSKeyCreation",
    "Effect": "Allow",
    "Action": [
        "kms:CreateKey",
        "kms:CreateAlias"
    ],
    "Resource": "*"
},
{
    "Sid": "KMSReadPermissions",
    "Effect": "Allow",
    "Action": [
        "kms:ListAliases"
    ],
    "Resource": "*"
},
{
    "Sid": "AllowUpdatingKeyPolicy",
    "Effect": "Allow",
    "Action": [
        "kms:PutKeyPolicy",
        "kms:GetKeyPolicy",
        "kms:ListKeyPolicies"
    ],
    "Resource": "arn:aws:kms:REGION:ACCOUNT_ID:key/*"
},
{
    "Sid": "AllowUseOfKMSKeyForS3",
    "Effect": "Allow",
    "Action": [
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:Decrypt",
```

```
            "kms:GenerateDataKey"
        ],
        "Resource": "arn:aws:kms:REGION:ACCOUNT_ID:key/*"
    },
    {
        "Sid": "AllowS3EncryptionConfiguration",
        "Effect": "Allow",
        "Action": [
            "s3:PutEncryptionConfiguration",
            "s3:GetEncryptionConfiguration"
        ],
        "Resource": "arn:aws:s3:::*rum*",
        "Condition": {
            "StringEquals": {
                "aws:ResourceAccount": "${aws:PrincipalAccount}"
            }
        }
    }
```

Other users who need to view CloudWatch RUM data but don't need to create CloudWatch RUM resources, can be granted the **AmazonCloudWatchRUMReadOnlyAccess** policy.

Set up a web application to use CloudWatch RUM

Use the steps in these sections to set up your web application to begin using CloudWatch RUM to collect performance data from real user sessions.

Topics

- Authorize your web application to send data to AWS
- Creating a CloudWatch RUM app monitor for a web application
- Modifying the code snippet to configure the CloudWatch RUM web client (optional)
- Inserting the CloudWatch app monitor code snippet into your application
- Testing your CloudWatch app monitor setup by generating user events

Authorize your web application to send data to AWS

You have four options to set up data authentication:

- Use Amazon Cognito and let CloudWatch RUM create a new Amazon Cognito identity pool for the application. This method requires the least effort to set up.

  The identity pool will contain an unauthenticated identity. This allows the CloudWatch RUM web client to send data to CloudWatch RUM without authenticating the user of the application.

  The Amazon Cognito identity pool has an attached IAM role. The Amazon Cognito unauthenticated identity allows the web client to assume the IAM role that is authorized to send data to CloudWatch RUM.
- Use Amazon Cognito for authentication. If you use this, you can use an existing Amazon Cognito identity pool or create a new one to use with this app monitor. If you use an existing identity pool, you must also modify the IAM role that is attached to the identity pool. Use this option for identity pools that support unauthenticated users. You can use identity pools only from the same Region.
- Use authentication from an existing identity provider that you have already set up. In this case, you must get credentials from the identity provider and your application must forward these credentials to the RUM web client.

  Use this option for identity pools that support only authenticated users.
- Use resource-based policies to manage access to your app monitor. This includes the ability to send unauthenticated requests to CloudWatch RUM without AWS credentials. To learn more about resource based policies and RUM, see Using resource-based policies with CloudWatch RUM.

The following sections include more details about these options.

Use an existing Amazon Cognito identity pool

If you choose to use a Amazon Cognito identity pool, you specify the identity pool when you add the application to CloudWatch RUM. The pool must support enabling access to unauthenticated identities. You can use identity pools only from the same Region.

You also must add the following permissions to the IAM policy that is attached to the IAM role that is associated with this identity pool.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rum:PutRumEvents"
            ],
            "Resource": "arn:aws:rum:us-east-1:123456789012:appmonitor/app
 monitor name"
        }
    ]
}
```

Amazon Cognito will then send the necessary security token to enable your application to access CloudWatch RUM.

Third-party provider

If you choose to use private authentication from a third-party provider, you must get credentials from the identity provider and forward them to AWS. The best way to do this is by using a *security token vendor*. You can use any security token vendor, including Amazon Cognito with AWS Security Token Service. For more information about AWS STS, see Welcome to the AWS Security Token Service API Reference.

If you want to use Amazon Cognito as the token vendor in this scenario, you can configure Amazon Cognito to work with an authentication provider. For more information, see Getting Started with Amazon Cognito Identity Pools (Federated Identities).

After you configure Amazon Cognito to work with your identity provider, you also need to do the following:

- Create an IAM role with the following permissions. Your application will use this role to access AWS.

  JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
```

```
      "Action": "rum:PutRumEvents",
      "Resource": "arn:aws:rum:us-east-2:123456789012:appmonitor/AppMonitorName"
    }
  ]
}
```

- Add the following to your application to have it pass the credentials from your provider to CloudWatch RUM. Insert the line so that it runs after a user has signed in to your application and the application has received the credentials to use to access AWS.

```
cwr('setAwsCredentials', {/* Credentials or CredentialProvider */});
```

For more information about credential providers in the AWS JavaScript SDK, see Setting credentials in a web browser in the v3 developer guide for SDK for JavaScript, Setting credentials in a web browser in the v2 developer guide for SDK for JavaScript, , and @aws-sdk/credential-providers.

You can also use the SDK for the CloudWatch RUM web client to configure the web client authentication methods. For more information about the web client SDK, see CloudWatch RUM web client SDK.

Creating a CloudWatch RUM app monitor for a web application

To start using CloudWatch RUM with your application, you create an *app monitor*. When the app monitor is created, RUM generates a code snippet for you to paste into your application. The snippet pulls in the RUM client code. The RUM client captures data from your application's user sessions and sends it to RUM.

To create an app monitor for a web platform

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose **Add app monitor**.
4. For **App monitor name**, enter a name to be used to identify this app monitor within the CloudWatch RUM console.
5. Select **Web** as the platform.
6. For **Application domain list**, enter the registered domain names where your application has administrative authority. You can also use a wildcard character * to allow any sub-domain or top-level domains (for example, *.amazon.com, amazon.*, *.amazon.*).

7. For **Configure RUM data collection**, specify whether you want the app monitor to collect each of the following:

   - **Performance telemetry** – Collects information about page load and resource load times
   - **JavaScript errors** – Collects information about unhandled JavaScript errors raised by your application

     You can select **Unminify JavaScript error stack traces** to debug unminified JavaScript errors. To use this feature, upload your source map files to an Amazon S3 bucket or folder and provide the Amazon S3 URI. Once enabled, RUM will use these source maps and enrich JavaScript error events by adding the unminified stack trace. Note that after enabling, this feature only processes new JavaScript error events and cannot be used on previously collected data. For more information, see Enabling unminification of JavaScript error stack traces.
   - **HTTP errors** – Collects information about HTTP errors thrown by your application

   Selecting these options provides more information about your application, but also generates more CloudWatch RUM events and thus incurs more charges.

   If you don't select any of these, the app monitor still collects session start events and page IDs so that you can see how many users are using your application, including breakdowns by operating system type and version, browser type and version, device type, and location.
8. Select **Check this option to allow the CloudWatch RUM Web Client to set cookies** if you want to be able to collect user IDs and session IDs from sampled user sessions. The user IDs are randomly generated by RUM. For more information, see CloudWatch RUM web client cookies (or similar technologies).
9. For **Session samples**, enter the percentage of user sessions that will be used to gather RUM data. The default is 100%. Reducing this number gives you less data, but reduces your charges. For more information about RUM pricing, see RUM pricing.
10. End user data that you collect for CloudWatch RUM is retained for 30 days and then deleted. If you want to keep copies of RUM events in CloudWatch Logs and configure how long to retain these copies, choose **Check this option to store your application telemetry data in your CloudWatch Logs account** under **Data storage**. By default, the CloudWatch Logs log group retains the data for 30 days. You can adjust the retention period in the CloudWatch Logs console.

11. (Optional) Choose to add a resource-based policy to your app monitor to control who can send `PutRumEvents` requests to your app monitor. If you choose **Create public policy**, a resource policy will be attached to your app monitor that enables anyone to send `PutRumEvents` requests to your app monitor. For more information about this method, see Using resource-based policies with CloudWatch RUM.
12. If you attached a resource-based policy in the previous step, then you don't need to sign requests to CloudWatch RUM with AWS credentials, and you can skip setting up authorization. Otherwise, for **Authorization**, specify whether to use a new or existing Amazon Cognito identity pool or use a different identity provider. Creating a new identity pool is the simplest option that requires no other setup steps. For more information, see see Authorize your web application to send data to AWS.

    Creating a new Amazon Cognito identity pool requires administrative permissions. For more information, see IAM policies to use CloudWatch RUM.
13. (Optional) By default, when you add the RUM code snippet to your application, the web client injects the JavaScript tag to monitor usage into the HTML code of all pages of your application. To change this, choose **Configure pages** and then choose either **Include only these pages** or **Exclude these pages**. Then, specify the pages to include or exclude. To specify a page to include or exclude, enter its complete URLs. To specify additional pages, choose **Add URL**.
14. To enable AWS X-Ray tracing of the user sessions that are sampled by the app monitor, choose **Active tracing** and select **Trace my service with AWS X-Ray.**

    If you select this, `XMLHttpRequest` and `fetch` requests made during user sessions sampled by the app monitor are traced. You can then see traces and segments from these user sessions in the RUM dashboard, and the X-Ray trace map and trace details pages. These user sessions will also show up as client pages in Application Signals after you have enabled it for your application.

    By making additional configuration changes to the CloudWatch RUM web client, you can add an X-Ray trace header to HTTP requests to enable end-to-end tracing of user sessions through to downstream AWS managed services. For more information, see Enabling X-Ray end-to-end tracing.
15. (Optional) To add tags to the app monitor, choose **Tags**, **Add new tag**.

    Then, for **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**.

To add another tag, choose **Add new tag** again.

For more information, see Tagging AWS Resources.

16. Choose **Add app monitor**.
17. In the **Sample code** section, you can copy the code snippet to use to add to your application. We recommend that you choose **JavaScript** or **TypeScript** and use NPM to install the CloudWatch RUM web client as a JavaScript module.

    Alternatively, you can choose **HTML** to use a content delivery network (CDN) to install the CloudWatch RUM web client. The disadvantage of using a CDN is that the web client is often blocked by ad blockers.
18. Choose **Copy** or **Download**, and then choose **Done**.

Modifying the code snippet to configure the CloudWatch RUM web client (optional)

You can modify the code snippet before inserting it into your application, to activate or deactivate several options. For more information, see the CloudWatch RUM web client documentation.

There are four configuration options that you should definitely be aware of, as discussed in these sections.

Preventing the collection of resource URLs that might contain personal information

By default, the CloudWatch RUM web client is configured to record the URLs of resources downloaded by the application. These resources include HTML files, images, CSS files, JavaScript files, and so on. For some applications, URLs may contain personally identifiable information (PII).

If this is the case for your application, we strongly recommend that you disable the collection of resource URLs by setting `recordResourceUrl: false` in the code snippet configuration, before inserting it into your application.

Manually recording page views

By default, the web client records page views when the page first loads and when the browser's history API is called. The default page ID is `window.location.pathname`. However, in some cases you might want to override this behavior and instrument the application to record page views programmatically. Doing so gives you control over the page ID and when it is recorded. For

example, consider a web application that has a URI with a variable identifier, such as `/entity/123` or `/entity/456`. By default, CloudWatch RUM generates a page view event for each URI with a distinct page ID matching the pathname, but you might want to group them by the same page ID instead. To accomplish this, disable the web client's page view automation by using the `disableAutoPageView` configuration, and use the `recordPageView` command to set the desired page ID. For more information, see Application-specific Configurations on GitHub.

Embedded script example:

```
cwr('recordPageView', { pageId: 'entityPageId' });
```

JavaScript module example:

```
awsRum.recordPageView({ pageId: 'entityPageId' });
```

Enabling X-Ray end-to-end tracing

When you create the app monitor, selecting **Trace my service with AWS X-Ray** enables the tracing of `XMLHttpRequest` and `fetch` requests made during user sessions that are sampled by the app monitor. You can then see traces from these HTTP requests in the CloudWatch RUM dashboard, and the X-Ray Trace Map and Trace details pages.

By default, these client-side traces are not connected to downstream server-side traces. To connect client-side traces to server-side traces and enable end-to-end tracing, set the `addXRayTraceIdHeader` option to `true` in the web client. This causes the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests.

The following code block shows an example of adding client-side traces. Some configuration options are omitted from this sample for readibility.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            enableXRay: true,
```

```
            telemetries: [
                'errors',
                'performance',
                [ 'http', { addXRayTraceIdHeader: true } ]
            ]
        }
    );
</script>
```

> **Warning**
>
> Configuring the CloudWatch RUM web client to add an X-Ray trace header to HTTP requests can cause cross-origin resource sharing (CORS) to fail or invalidate the request's signature if the request is signed with SigV4. For more information, see the CloudWatch RUM web client documentation. We strongly recommend that you test your application before adding a client-side X-Ray trace header in a production environment.

For more information, see the CloudWatch RUM web client documentation

Sending unsigned requests to CloudWatch RUM

By default, the RUM web client signs all requests sent to RUM. If you set `signing:false` in the client configuration, requests will be unsigned when they are sent to CloudWatch RUM. Data will be ingested to RUM only if there is a public resource based policy attached to the app monitor. For more information, see Using resource-based policies with CloudWatch RUM.

Inserting the CloudWatch app monitor code snippet into your application

Next, you insert the code snippet that you created in the previous section into your application.

> **Warning**
>
> The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. Before you insert the code snippet, see Filtering by metadata attributes in the console.

If you don't have the code snippet that was previously generated, you can find it by following the directions in How do I find a code snippet that I've already generated?.

To insert the CloudWatch RUM code snippet into your application

1. Insert the code snippet that you copied or downloaded in the previous section inside the <head> element of your application. Insert it before the <body> element or any other <script> tags.

   The following is an example of a generated code snippet:

```
<script>
(function (n, i, v, r, s, c, x, z) {
    x = window.AwsRumClient = {q: [], n: n, i: i, v: v, r: r, c: c};
    window[n] = function (c, p) {
        x.q.push({c: c, p: p});
    };
    z = document.createElement('script');
    z.async = true;
    z.src = s;
    document.head.insertBefore(z, document.getElementsByTagName('script')[0]);
})('cwr',
    '194a1c89-87d8-41a3-9d1b-5c5cd3dafbd0',
    '1.0.0',
    'us-east-2',
    'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
    {
        sessionSampleRate: 1,
        identityPoolId: "us-east-2:c90ef0ac-e3b8-4d1a-b313-7e73cfd21443",
        endpoint: "https://dataplane.rum.us-east-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        enableXRay: false
    });
</script>
```

2. If your application is a multipage web application, you must repeat step 1 for each HTML page that you want included in the data collection.

Testing your CloudWatch app monitor setup by generating user events

After you have inserted the code snippet and your updated application is running, you can test it by manually generating user events. To test this, we recommend that you do the following. This testing incurs standard CloudWatch RUM charges.

- Navigate between pages in your web application.
- Create multiple user sessions, using different browsers and devices.
- Make requests.
- Cause JavaScript errors.

After you have generated some events, view them in the CloudWatch RUM dashboard. For more information, see Viewing the CloudWatch RUM dashboard.

Data from user sessions might take up to 15 minutes to appear in the dashboard.

If you don't see data 15 minutes after you generated events in the application, see Troubleshooting CloudWatch RUM.

Using resource-based policies with CloudWatch RUM

You can attach a resource policy to a CloudWatch RUM app monitor. By default, app monitors do not have a resource policy attached to them. CloudWatch RUM resource based policies do not support cross-account access.

To learn more about AWS resource policies, see Identity-based policies and resource-based policies.

To learn more about how resource policies and identity policies are evaluated, see Policy evaluation logic.

To learn more about IAM policy grammar, see IAM JSON policy element reference.

Supported actions

Resource-based policies on app monitors support the `rum:PutRumEvents` action.

Sample policies to use with CloudWatch RUM

The following example allows anyone to write data to your app monitor, including those without SigV4 credentials.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:us-east-1:123456789012:appmonitor/app
 monitor name",
            "Principal": "*"
        }
    ]
}
```

You can modify the policy to block specified source IP addresses by using the `aws:SourceIp` condition key. With this example, Using this policy, PutRumEvents from the IP address listed will be rejected. All other requests from other IP addresses will be accepted. For more information about this condition key, see Properties of the network in the IAM User Guide.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:us-
east-1:123456789012:appmonitor/AppMonitorName",
            "Principal": "*"
        },
        {
            "Effect": "Deny",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:us-
east-1:123456789012:appmonitor/AppMonitorName",
            "Principal": "*",
            "Condition": {
                "NotIpAddress": {
                "aws:SourceIp": "198.51.100.252"
                }
            }
        }
    ]
```

```
}
```

Additionally, you can use the `rum:alias` service context key to control which requests are accepted.

For web app monitors, you must configure your web client to send `Alias` using version 1.20 or later of the CloudWatch RUM web client, as described in Application-specific Configurations on GitHub.

For mobile app monitors, you must configure your instrumentation according to the SDK.

- iOS applications use the AWS Distro for OpenTelemetry (ADOT) iOS SDK.
- Android applications use the AWS Distro for OpenTelemetry (ADOT) Android SDK.

In the following example, the resource policy requires that requests must contain either `alias1` or `alias2` for the event to be accepted.

```
    {
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRUMPutEvents",
            "Effect": "Allow",
            "Action": "rum:PutRumEvents",
            "Resource": "arn:aws:rum:us-east-1:123456789012:appmonitor/MyApplication",
            "Principal": "*",
            "Condition": {
                "StringEquals": {
                    "rum:alias":["alias1", "alias2"]
                }
            }
        }
    ]
}
```

Configuring the CloudWatch RUM web client

Your applications can use one of the code snippets generated by CloudWatch RUM to install the CloudWatch RUM web client. The generated snippets support two installation methods: as a

JavaScript module through NPM, or from a content delivery network (CDN). For best performance, we recommend using the NPM installation method. For more information about using this method, see Installing as a JavaScript Module.

If you use the CDN installation option, ad blockers might block the default CDN provided by CloudWatch RUM. This disables application monitoring for users who have ad blockers installed. Because of this, we recommend that you use the default CDN only for initial onboarding with CloudWatch RUM. For more information about the ways to mitigate this issue, see Instrument the application.

The code snippet sits in the <head> tag of an HTML file and installs the web client by downloading the web client, and then configuring the web client for the application it is monitoring. The snippet is a self-executing function which looks similar to the following. In this example, the body of the snippet's function has been omitted for readability.

```
<script>
(function(n,i,v,r,s,c,u,x,z){...})(
'cwr',
'00000000-0000-0000-0000-000000000000',
'1.0.0',
'us-west-2',
'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
{ /* Configuration Options Here */ }
);
<script>
```

Arguments

The code snippet accepts six arguments:

- A namespace for running commands on the web client, such as `'cwr'`
- The ID of the app monitor, such as `'00000000-0000-0000-0000-000000000000'`
- The application version, such as `'1.0.0'`
- The AWS Region of the app monitor, such as `'us-west-2'`
- The URL of the web client, such as `'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js'`
- Application-specific configuration options. For more information, see the following section.

Ignoring errors

The CloudWatch RUM web client listens to all types of errors that happen in your applications. If your application emits JavaScript errors that you do not want to view in the CloudWatch RUM dashboard, you can configure the CloudWatch RUM web client to filter out these errors so that you see only the relevant error events on the CloudWatch RUM dashboard. For example, you might choose not to view some JavaScript errors in the dashboard because you have already identified a fix for them and the volume of these errors is masking other errors. You might also want to ignore errors that you can't fix because they are owned by a library owned by a third party.

For more information about how to instrument the web client to filter out specific JavaScript errors, see the example in Errors in the web client Github documentation.

Configuration options

For information about the configuration options available for the CloudWatch RUM web client, see the CloudWatch RUM web client documentation

Session replay

CloudWatch RUM session replay allows you to capture end user sessions for your web application. You can play back these sessions to visually see what users experienced, helping you identify issues, understand user behavior, and troubleshoot problems.

To protect user privacy, all text input and text displayed on the page is masked by default. This means that sensitive information such as names, addresses, and other personal data is not recorded.

Enabling session replay

To use session replay, import the `RRWebPlugin` from the `aws-rum-web` package and add it to the `eventPluginsToLoad` array in your web client configuration. For more information about installing the web client as a JavaScript module, see Configuring the CloudWatch RUM web client.

The following example shows how to enable session replay.

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';
import { RRWebPlugin } from 'aws-rum-web';

const config: AwsRumConfig = {
```

```
    identityPoolId: 'us-west-2:00000000-0000-0000-0000-000000000000',
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance', 'http'],
    eventPluginsToLoad: [new RRWebPlugin()]
};

const awsRum: AwsRum = new AwsRum(
    'APPLICATION_ID',
    '1.0.0',
    'us-west-2',
    config
);
```

The `RRWebPlugin` accepts optional configuration options, such as sampling rate and recording behavior. For the full list of available configuration options, see the CloudWatch RUM web client configuration on GitHub.

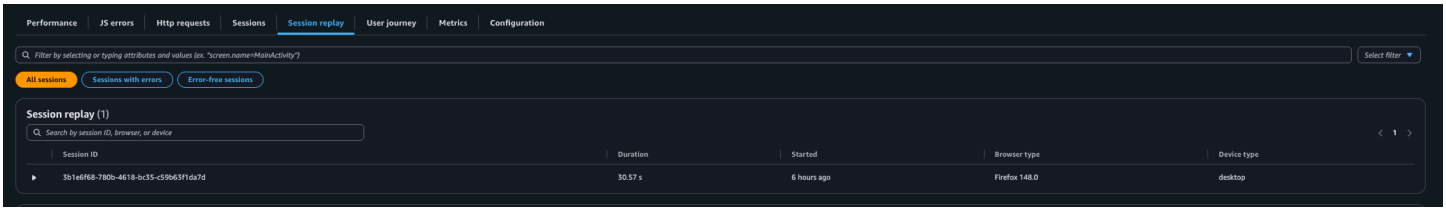
Privacy and data masking

Session replay masks all text input and text content on the page by default. This includes form fields, labels, paragraphs, and any other text rendered in the DOM. Masked content appears as placeholder characters during playback, ensuring that personally identifiable information (PII) is not captured or stored.

Viewing session replays

After you enable session replay, you can view recorded sessions in the CloudWatch RUM console. Navigate to your app monitor and choose the **Session replay** tab. This tab displays a list of recorded sessions with details including session ID, duration, start time, browser type, and device type. You can filter sessions to show all sessions, sessions with errors, or error-free sessions.

You can also find sessions with replays in the **Sessions** tab. Any session that has a replay available displays a play button next to it. Choose the play button to open the replay player.



Choose a session to open the replay player. The player shows a visual playback of the user's session, including a timeline of interactions on the right side. The timeline displays each user interaction,

such as page navigations, scrolls, clicks, and text inputs, along with the timestamp when each interaction occurred. You can use the playback controls to adjust the speed (1x, 2x, 4x, or 8x) and skip inactive periods.



Enabling unminification of JavaScript error stack traces

When your web application JavaScript source code is minified, error stack traces can be difficult to read. You can enable unminification to the stack traces by uploading your source maps to Amazon S3. CloudWatch RUM will retrieve the source maps to map the line and column numbers in the minified source code back to the original unminified source code. This will improve readability of your error stack traces and help identify the location of the error in the original source code.

Requirements and syntax

Source maps are crucial for debugging and tracking issues in your web application across different releases. Make sure that each web application release has a unique source map. Each release should have its own unique releaseId. A releaseId must be a string between 1 and 200 characters long and can only contain letters, numbers, underscores, hyphens, colons, forward slashes, and periods. To add the `releaseId` as metadata to RUM events, configure the CloudWatch RUM web client.

Source maps are expected to be plain JSON files following the structure defined by the Source Map V3 specification. The required fields are: `version`, `file`, `sources`, `names`, and `mappings`.

Make sure the size of each source map does not exceed the limit of 50 MB. In addition, RUM service will only retrieve up to 50 MB of source maps per stack trace. If needed, split the source code into multiple smaller chunks. For more information, see Code Splitting with WebpackJS.

Topics

- Configure your Amazon S3 bucket resource policy to allow RUM service access
- Upload source maps
- Configure releaseId in your CloudWatch RUM web client
- Enabling CloudWatch RUM app monitor to unminify JavaScript stack traces
- Viewing unminified stack traces in the RUM console
- Viewing unminified stack traces in CloudWatch Logs
- Troubleshooting source maps

Configure your Amazon S3 bucket resource policy to allow RUM service access

Make sure your Amazon S3 bucket is in the same region as your RUM appMonitor. Configure your Amazon S3 bucket to allow RUM service access for retrieving source map files. Include the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to limit the service's permissions to the resource. This is the most effective way to protect against the confused deputy problem.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon S3 to prevent the confused deputy problem.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "RUM Service S3 Read Permissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "rum.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
```

```
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::BUCKET_NAME",
                "arn:aws:s3:::BUCKET_NAME/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "ACCOUNT_ID",
                    "aws:SourceArn":
 "arn:aws:rum:REGION:ACCOUNT_ID:appmonitor/APP_MONITOR_NAME"
                }
            }
        }
    ]
}
```

If you are using AWS KMS keys to encrypt the data, make sure the key's resource policy is configured similarly to include the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to give RUM service access to use the keys to retrieve the source map files.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "RUM Service KMS Read Permissions",
            "Effect": "Allow",
            "Principal": {
                "Service": "rum.amazonaws.com"
            },
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-1:123456789012:key/KEY_ID",
            "Condition": {
                "StringEquals": {
                "aws:SourceAccount": "123456789012",
    "aws:SourceArn": "arn:aws:rum:us-east-1:123456789012/APP_MONITOR_NAME"
                }
            }
        }
```

```
    ]
}
```

Upload source maps

Configure your JavaScript bundle to generate source maps during minification. When you build your application, the bundle will create a directory (for example, dist) containing the minified JavaScript files and their corresponding source maps. See below for an example.

```
./dist
    |-index.d5a07c87.js
    |-index.d5a07c87.js.map
```

Upload the source map files to your Amazon S3 bucket. The files should be located in a folder with the `releaseId` as the name. For example, if my bucket name is `my-application-source-maps` and the `releaseId` is 2.0.0, then the source map file is located at the following location:

```
my-application-source-maps
    |-2.0.0
        |-index.d5a07c87.js.map
```

To automate uploading your source maps, you can create the following bash script and execute it as part of your build process.

```
#!/bin/bash
# Ensure the script is called with required arguments
if [ "$#" -ne 2 ]; then
 echo "Usage: $0 S3_BUCKET_NAME RELEASE_ID"
 exit 1
fi

# Read arguments
S3_BUCKET="$1"
RELEASE_ID="$2"

# Set the path to your build directory
BUILD_DIR="./dist"


# Upload all .map files recursively
```

```
 if aws s3 cp "$BUILD_DIR" "s3://$S3_BUCKET/$RELEASE_ID/" --recursive --exclude "*" --
include "*.map"; then
    echo "Successfully uploaded all source map files"
else
    echo "Failed to upload source map files"
fi
```

Configure releaseId in your CloudWatch RUM web client

CloudWatch RUM uses the configured `releaseId` to determine the folder to retrieve the source map files. Name the `releaseId` the same as your source map files folder. If you used the provided bash script above or a similar one, the `releaseId` configured in the script should be the same as the one configured in your CloudWatch RUM web client. You must use version 1.21.0 or later of the CloudWatch RUM web client.

```
import { AwsRum, AwsRumConfig } from "aws-rum-web";

try {
    const config: AwsRumConfig = {
        sessionSampleRate: 1,
        endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
        telemetries: ["performance", "errors", "http"],
        allowCookies: true,
        releaseId: "RELEASE_ID", //Add this
    };

    const APPLICATION_ID: string = "APP_MONITOR_ID";
    const APPLICATION_VERSION: string = "1.0.0";
    const APPLICATION_REGION: string = "us-west-2";

    new AwsRum(APPLICATION_ID, APPLICATION_VERSION, APPLICATION_REGION, config);
} catch (error: any) {
    // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

Enabling CloudWatch RUM app monitor to unminify JavaScript stack traces

To unminify JavaScript stack traces, set the app monitor's SourceMap status to ENABLED. Provide the Amazon S3 URI to the bucket or folder containing all source maps for your app monitor.

When storing source maps directly in the main bucket (not in a subfolder), then the Amazon S3 URI should be formatted as `Amazon S3://BUCKET_NAME`. In this case, source map files should be located at the following location.

```
BUCKET_NAME
    |- RELEASE_ID
        |-index.d5a07c87.js.map
```

When a child directory is the root, then the Amazon S3 URI should be formatted as `Amazon S3://BUCKET_NAME/DIRECTORY`. In this case, source map files should be located at the following location.

```
BUCKET_NAME
    |- DIRECTORY
        |-RELEASE_ID
            |-index.d5a07c87.js.map
```

Viewing unminified stack traces in the RUM console

After uploading your source maps to Amazon S3, enabling source maps on your RUM app monitor, and deploying your web application with the `releaseId` configured in the CloudWatch RUM web client, select **Events** in the RUM console. This tab displays the raw RUM event data. Filter by the JS error event type and view the latest JS error event. You will see the unminified stack trace in the new `event_details.unminifiedStack` field for events ingested after the feature was enabled.

Viewing unminified stack traces in CloudWatch Logs

Enable RUM event storage in CloudWatch Logs by turning on **Data storage**. Once enabled, you can search the new **event_details.unminifiedStack** field. This allows you to analyze trends and relate issues across multiple sessions using CloudWatch Logs queries.

Troubleshooting source maps

CloudWatch RUM provides out of the box metrics to troubleshoot your source map setup. These metrics are published in the metric namespace named `AWS/RUM`. The following metrics are published with an application_name dimension. The value of this dimension is the name of the app monitor. The metrics are also published with an `aws:releaseId` dimension. The value of this dimension is the `releaseId` associated with the JavaScript error event.

MetricName	Unit	Description
UnminifyLineFailureCount	Count	The count of stack trace lines in the JS error event that failed to be unminified. Additional details regarding the failure will be added to the specific line that failed in the event_details.unminifiedStack field.
UnminifyLineSuccessCount	Count	The count of stack trace lines in the JS error event that were successfully unminified.
UnminifyEventFailureCount	Count	The count of JS error events that failed to have any lines unminified. Additional details regarding the failure will be added in the event_details.unminifiedStack field.
UnminifyEventSuccessCount	Count	The count of JS error events that succeeded to have at least one stack trace line unminified.

CloudWatch RUM may fail to unminify a line in the stack trace for various reasons, including but not limited to:

- Failure to retrieve corresponding source map file due to permission issues. Make sure the bucket resource policy is configured correctly.
- Corresponding source map file does not exist. Make sure the source map files have been uploaded to the correct bucket or folder that has the same name as the releaseId configured in your CloudWatch RUM web client.
- Corresponding source map file is too big. Split your source code into smaller chunks.

- 50 MB worth of source map files already retrieved for the stack trace. Reduce the stack trace length as 50 MB is service side limitation.
- Source map is invalid and could not be indexed. Make sure the source map is a plain JSON following the structure defined by the Source Map V3 specification and includes the following fields: version, file, sources, names, mappings.
- Source map could not map the minified source code back to the unminified stack trace. Make sure the source map is the correct source map for the given releaseId.

Regionalization

This section illustrates strategies for using CloudWatch RUM with applications in different Regions.

My application is deployed in multiple AWS Regions

If your application is deployed in multiple AWS Regions, you have three options:

- Deploy one app monitor in one Region, in one account, serving all Regions.
- Deploy separate app monitors for each Region, in unique accounts.
- Deploy separate app monitors for each Region, all in one account.

The advantage of using one app monitor is that all data will be centralized into one visualization, and all logs are written to the same log group in CloudWatch Logs. With a single app monitor there is a small amount of extra latency for requests, and a single point of failure.

Using multiple app monitors removes the single point of failure, but prevents all data from being combined into one visualization.

CloudWatch RUM hasn't launched in some Regions that my application is deployed in

CloudWatch RUM is launched into many Regions and has wide geographical coverage. By setting up CloudWatch RUM in the Regions where it is available, you can get the benefits. End users can be anywhere and still have their sessions included if you have set up an app monitor in the Region that they are connecting to.

However, CloudWatch RUM is not yet launched in any Regions in China. You are not able to send data to CloudWatch RUM from these Regions.

Use page groups

Use page groups to associate different pages in your application with each other so that you can see aggregated analytics for groups of pages. For example, you might want to see the aggregated page load times of all of your landing pages.

You put pages into page groups by adding one or more tags to page view events in the CloudWatch RUM web client. The following examples put the `/home` page into the page group named `en` and the page group named `landing`.

Embedded script example

```
cwr('recordPageView', { pageId: '/home', pageTags: ['en', 'landing']});
```

JavaScript module example

```
awsRum.recordPageView({ pageId: '/home', pageTags: ['en', 'landing']});
```

> **Note**
>
> Page groups are intended to facilitate aggregating analytics across different pages. For information about how to define and manipulate `pageIds` for your application, see the **Manually recording page views** section in Modifying the code snippet to configure the CloudWatch RUM web client (optional).

Specify custom metadata

CloudWatch RUM attaches additional data to each event as metadata. Event metadata consists of attributes in the form of key-value pairs. You can use these attributes to search or filter events in the CloudWatch RUM console. By default, CloudWatch RUM creates some metadata for you. For more information about the default metadata, see RUM event metadata.

You can also use the CloudWatch RUM web client to add custom metadata to CloudWatch RUM events. The custom metadata can include session attributes and page attributes.

To add custom metadata, you must use version 1.10.0 or later of the CloudWatch RUM web client.

Requirements and syntax

Each event can include as many as 10 custom attributes in the metadata. The syntax requirements for custom attributes are as follows:

- **Keys**
  - Maximum of 128 characters
  - Can include alphanumeric characters, colons (:), and underscores (_)
  - Can't begin with `aws:`.
  - Can't consist entirely of any of the reserved keywords listed in the following section. Can use those keywords as part of a longer key name.
- **Values**
  - Maximum of 256 characters
  - Must be strings, numbers, or Boolean values

Reserved keywords

You can't use the following reserved keywords as complete key names. You can use the following keywords as part of a longer key name, such as `applicationVersion`.

- `browserLanguage`
- `browserName`
- `browserVersion`
- `countryCode`
- `deviceType`
- `domain`
- `interaction`
- `osName`
- `osVersion`
- `pageId`
- `pageTags`
- `pageTitle`
- `pageUrl`

- `parentPageId`
- `platformType`
- `referrerUrl`
- `subdivisionCode`
- `title`
- `url`
- `version`

> **Note**
>
> CloudWatch RUM removes custom attributes from RUM events if an attribute includes a key or value that is not valid, or if the limit of 10 custom attributes per event has already been reached.

Add session attributes

If you configure custom session attributes, they are added to all events in a session. You configure session attributes either during CloudWatch RUM web client initialization or at runtime by using the `addSessionAttributes` command.

For example, you can add your application's version as a session attribute. Then, in the CloudWatch RUM console, you can filter errors by version to find whether an increased error rate is associated with a particular version of your application.

Adding a session attribute at initialization, NPM example

The code section in bold adds the session attribute.

```
import { AwsRum, AwsRumConfig } from 'aws-rum-web';

try {
  const config: AwsRumConfig = {
    allowCookies: true,
    endpoint: "https://dataplane.rum.us-west-2.amazonaws.com",
    guestRoleArn: "arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth",
```

```
    identityPoolId: "us-west-2:00000000-0000-0000-0000-000000000000",
    sessionSampleRate: 1,
    telemetries: ['errors', 'performance'],
    sessionAttributes: {
        applicationVersion: "1.3.8"
    }
  };

  const APPLICATION_ID: string = '00000000-0000-0000-0000-000000000000';
  const APPLICATION_VERSION: string = '1.0.0';
  const APPLICATION_REGION: string = 'us-west-2';

  const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    config
  );
} catch (error) {
  // Ignore errors thrown during CloudWatch RUM web client initialization
}
```

Adding a session attribute at runtime, NPM example

```
awsRum.addSessionAttributes({
    applicationVersion: "1.3.8"
})
```

Adding a session attribute at initialization, embedded script example

The code section in bold adds the session attribute.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            sessionSampleRate:1,
            guestRoleArn:'arn:aws:iam::000000000000:role/RUM-Monitor-us-
west-2-000000000000-00xx-Unauth',
```

```
            identityPoolId:'us-west-2:00000000-0000-0000-0000-000000000000',
            endpoint:'https://dataplane.rum.us-west-2.amazonaws.com',
            telemetries:['errors','http','performance'],
            allowCookies:true,
            sessionAttributes: {
                applicationVersion: "1.3.8"
            }
        }
    );
</script>
```

Adding a session attribute at runtime, embedded script example

```
<script>
    function addSessionAttribute() {
        cwr('addSessionAttributes', {
            applicationVersion: "1.3.8"
        })
    }

</script>
```

Add page attributes

If you configure custom page attributes, they are added to all events on the current page. You configure page attributes either during CloudWatch RUM web client initialization or at runtime by using the `recordPageView` command.

For example, you can add your page template as a page attribute. Then, in the CloudWatch RUM console, you can filter errors by page templates to find whether an increased error rate is associated with a particular page template of your application.

Adding a page attribute at initialization, NPM example

The code section in bold adds the page attribute.

```
const awsRum: AwsRum = new AwsRum(
    APPLICATION_ID,
    APPLICATION_VERSION,
    APPLICATION_REGION,
    { disableAutoPageView:  true // optional }
);
```

```
awsRum.recordPageView({
    pageId:'/home',
    pageAttributes: {
      template: 'artStudio'
    }
});
const credentialProvider = new CustomCredentialProvider();
if(awsCreds) awsRum.setAwsCredentials(credentialProvider);
```

Adding a page attribute at runtime, NPM example

```
awsRum.recordPageView({
    pageId: '/home',
    pageAttributes: {
        template: 'artStudio'
    }
});
```

Adding a page attribute at initialization, embedded script example

The code section in bold adds the page attribute.

```
<script>
    (function(n,i,v,r,s,c,u,x,z){...})(
        'cwr',
        '00000000-0000-0000-0000-000000000000',
        '1.0.0',
        'us-west-2',
        'https://client.rum.us-east-1.amazonaws.com/1.0.2/cwr.js',
        {
            disableAutoPageView: true //optional
        }
    );
    cwr('recordPageView', {
       pageId: '/home',
       pageAttributes: {
           template: 'artStudio'
       }
    });
    const awsCreds = localStorage.getItem('customAwsCreds');
    if(awsCreds) cwr('setAwsCredentials', awsCreds)
</script>
```

Adding a page attribute at runtime, embedded script example

```
<script>
    function recordPageView() {
        cwr('recordPageView', {
            pageId: '/home',
            pageAttributes: {
                template: 'artStudio'
            }
        });
    }
</script>
```

Filtering by metadata attributes in the console

To filter the visualizations in the CloudWatch RUM console with any built-in or custom metadata attribute, use the search bar. In the search bar, you can specify as many as 20 filter terms in the form of **key=value** to apply to the visualizations. For example, to filter data for only the Chrome browser, you could add the filter term **browserName=Chrome**.

By default, the CloudWatch RUM console retrieves the 100 most common attributes keys and values to display in the dropdown in the search bar. To add more metadata attributes as filter terms, enter the complete attribute key and value into the search bar.

A filter can include as many as 20 filter terms, and you can save up to 20 filters per app monitor. When you save a filter, it is saved in the **Saved filters** dropdown. You can also delete a saved filter.

Send custom events

CloudWatch RUM records and ingests the events listed in Information collected by the CloudWatch RUM web client. If you use version 1.12.0 or later of the CloudWatch RUM web client, you can define, record, and send additional custom events. You define the event type name and the data to send for each event type that you define. Each custom event payload can be up to 6 KB.

Custom events are ingested only if the app monitor has custom events enabled. To update the configuration settings of your app monitor, use the CloudWatch RUM console or the UpdateAppMonitor API.

After you enable custom events, and then define and send custom events, you can search for them. To search for them, use the **Events** tab in the CloudWatch RUM console. Search by using the event type.

Requirements and syntax

Custom events consist of an event type and event details. The requirements for these are as follows:

- **Event type**
  - This can be either the **type** or **name** of your event. For example, the CloudWatch RUM built-in event type called **JsError** has an event type of `com.amazon.rum.js_error_event`.
  - Must be between 1 and 256 characters.
  - Can be a combination of alphanumeric characters, underscores, hyphens, and periods.
- **Event details**
  - Contains the actual data that you want to record in CloudWatch RUM.
  - Must be an object that consists of fields and values.

Examples of recording custom events

There are two ways to record custom events in the CloudWatch RUM web client.

- Use the CloudWatch RUM web client's `recordEvent` API.
- Use a customized plugin.

Send a custom event using the `recordEvent` API, NPM example

```
awsRum.recordEvent('my_custom_event', {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
)
```

Send a custom event using the `recordEvent` API, embedded script example

```
cwr('recordEvent', {
    type: 'my_custom_event',
```

```
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

Example of sending a custom event using a customized plugin

```
// Example of a plugin that listens to a scroll event, and
// records a 'custom_scroll_event' that contains the timestamp of the event.
class MyCustomPlugin implements Plugin {
    // Initialize MyCustomPlugin.
    constructor() {
        this.enabled;
        this.context;
        this.id = 'custom_event_plugin';
    }
    // Load MyCustomPlugin.
    load(context) {
        this.context = context;
        this.enable();
    }
    // Turn on MyCustomPlugin.
    enable() {
        this.enabled = true;
        this.addEventHandler();
    }
    // Turn off MyCustomPlugin.
    disable() {
        this.enabled = false;
        this.removeEventHandler();
    }
    // Return MyCustomPlugin Id.
    getPluginId() {
        return this.id;
    }
    // Record custom event.
    record(data) {
```

```
        this.context.record('custom_scroll_event', data);
    }
    // EventHandler.
    private eventHandler = (scrollEvent: Event) => {
        this.record({timestamp: Date.now()})
    }
    // Attach an eventHandler to scroll event.
    private addEventHandler(): void {
        window.addEventListener('scroll', this.eventHandler);
    }
    // Detach eventHandler from scroll event.
    private removeEventHandler(): void {
        window.removeEventListender('scroll', this.eventHandler);
    }
}
```

Viewing the CloudWatch RUM dashboard

CloudWatch RUM collects and visualizes application performance data from user sessions through an interactive dashboard. By capturing load times, Apdex scores, device information, geolocation, and error patterns, teams can quickly identify performance bottlenecks, prioritize fixes based on real user impact, and make sure optimal experiences across different browsers, devices, and geographic regions—helping organizations better understand user behavior leading to reduced end user frustration and improving application reliability.

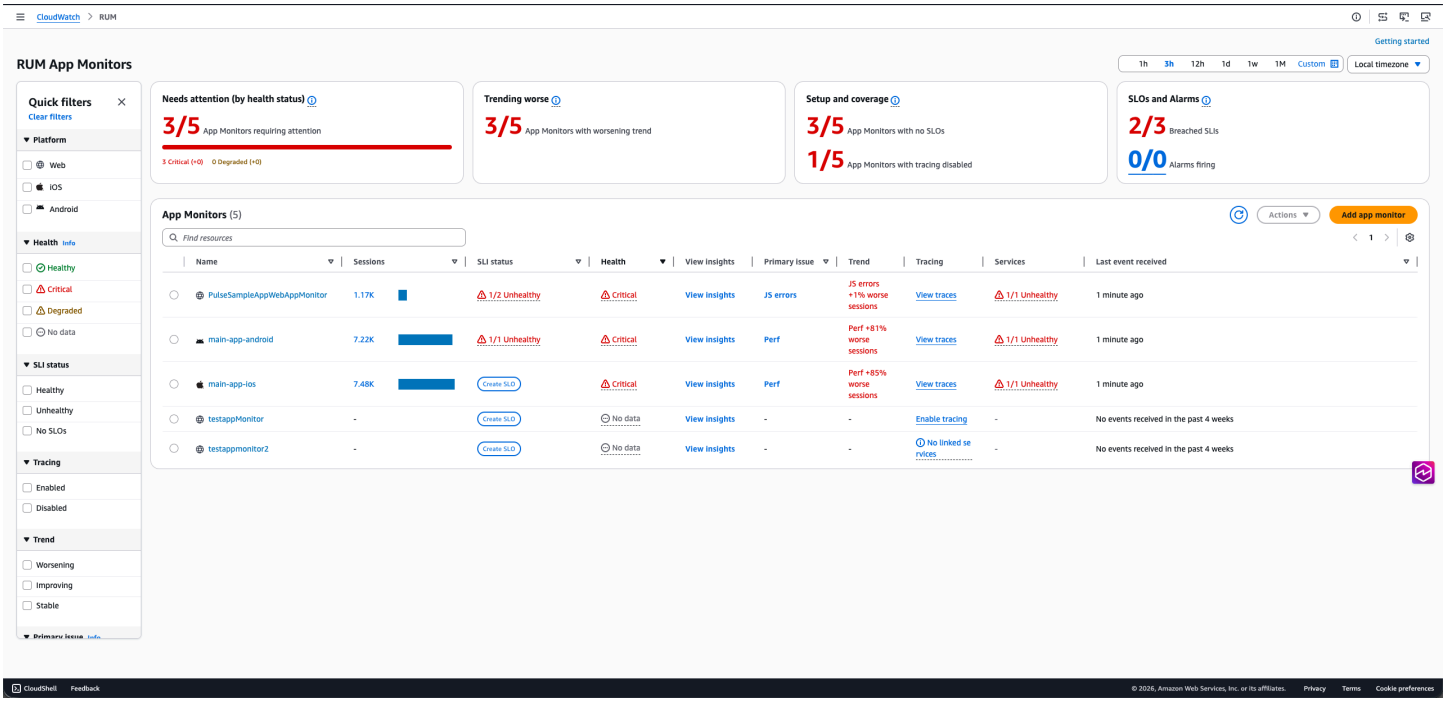
Getting Started with CloudWatch RUM

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals (APM)**, **RUM**.

The RUM console displays the **Overview** page, which provides a consolidated view of all your app monitors, their health, and key operational metrics. From the Overview page, select an app monitor to access detailed views with **Performance**, **Errors**, **Sessions**, **Metrics**, and **Configuration** tabs.

Overview

The Overview page is the landing page of the CloudWatch RUM console. It provides a high-level summary of all your application monitors, helping you quickly assess health, performance trends, and operational coverage across your monitored applications.



Summary cards

At the top of the Overview page, four summary cards provide an at-a-glance operational status across all your app monitors:

- **Needs attention (by health status)** — Shows how many app monitors require attention out of the total, broken down by **Critical** and **Degraded** counts. A progress bar indicates the proportion of monitors needing attention.
- **Trending worse** — Shows how many app monitors have a worsening trend out of the total.
- **Setup and coverage** — Shows how many app monitors have no SLOs configured and how many have tracing disabled, helping you identify gaps in your monitoring setup.
- **SLOs and Alarms** — Shows the number of breached SLIs out of total SLIs, and the number of alarms currently firing.

Quick filters

The left panel provides quick filters to narrow the application list by:

- **Platform** — Web, iOS, or Android.
- **Health** — Healthy, Critical, Degraded, or No data.
- **SLI status** — Healthy, Unhealthy, or No SLOs.

- **Tracing** — Enabled or Disabled.
- **Trend** — Worsening, Improving, or Stable.
- **Primary issue** — Filter by the primary issue type affecting the app monitor.

Choose **Clear filters** to reset all filters.

App Monitors table

The **App Monitors** table lists all your app monitors with the following columns:

- **Name** — The name of the app monitor, with a platform icon (Web, iOS, or Android).
- **Sessions** — The number of sessions recorded in the selected time range, displayed with a bar chart visualization.
- **SLI status** — The status of service level indicators. Displays the count of unhealthy SLIs (for example, "1/2 Unhealthy") or a **Create SLO** link if no SLOs are configured.
- **Health** — The health status of the application: **Healthy**, **Critical**, **Degraded**, or **No data**.
- **View Insights** — Choose this link to open the diagnostic side panel for the app monitor (see Diagnostic side panel).
- **Primary issue** — The primary issue type affecting the application. For **web** app monitors, values include **JS errors**, **Perf**, or **HTTP errors/faults**. For **mobile** app monitors, values include **Crashes**, **ANRs/App Hangs**, **Perf**, or **HTTP errors/faults**.
- **Trend** — A description of the trend direction and magnitude (for example, "JS errors +1% worse sessions" or "Perf +81% worse sessions").
- **Tracing** — A **View traces** link if tracing is enabled, or an **Enable tracing** link if it is not.
- **Services** — The SLI health status for linked services (for example, "1/1 Unhealthy"), or a dash if no services are linked.
- **Last event received** — The time since the last telemetry event was received (for example, "1 minute ago" or "No events received in the past 4 weeks").

Use the search bar above the table to find specific app monitors by name. You can sort the table by clicking column headers, and use the gear icon to customize which columns are visible. The **Actions** dropdown and **Add app monitor** button let you manage your app monitors directly from this page.

Health status

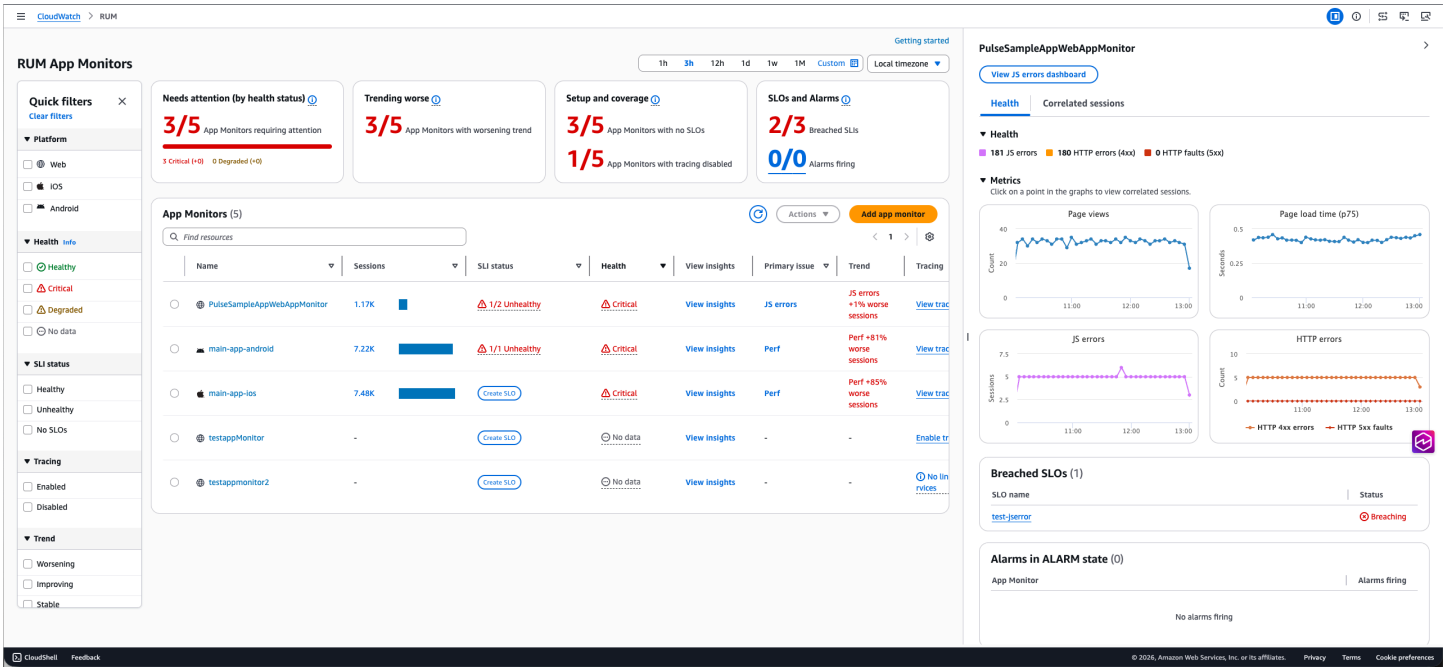
The **Health** column provides an at-a-glance assessment of each application's operational state based on the percentage of impacted sessions—sessions with errors or slow page loads—relative to total sessions in the selected time range.

Each app monitor displays one of the following statuses:

Status	Impacted sessions	Description
Healthy	Less than 1%	The application is operating within normal parameters.
Degraded	Between 1% and 5%	The application is experiencing elevated error rates or degraded performance that may require attention.
Critical	More than 5%	The application is experiencing significant errors or performance issues that require immediate investiga tion.
No data	—	Insufficient data is available to determine the applicati on's health status. This can occur when an app monitor has not received recent telemetry data.

Diagnostic side panel

When you choose **View Insights** for an app monitor in the table, a diagnostic side panel opens on the right side of the page. The panel displays the app monitor name and provides two tabs:



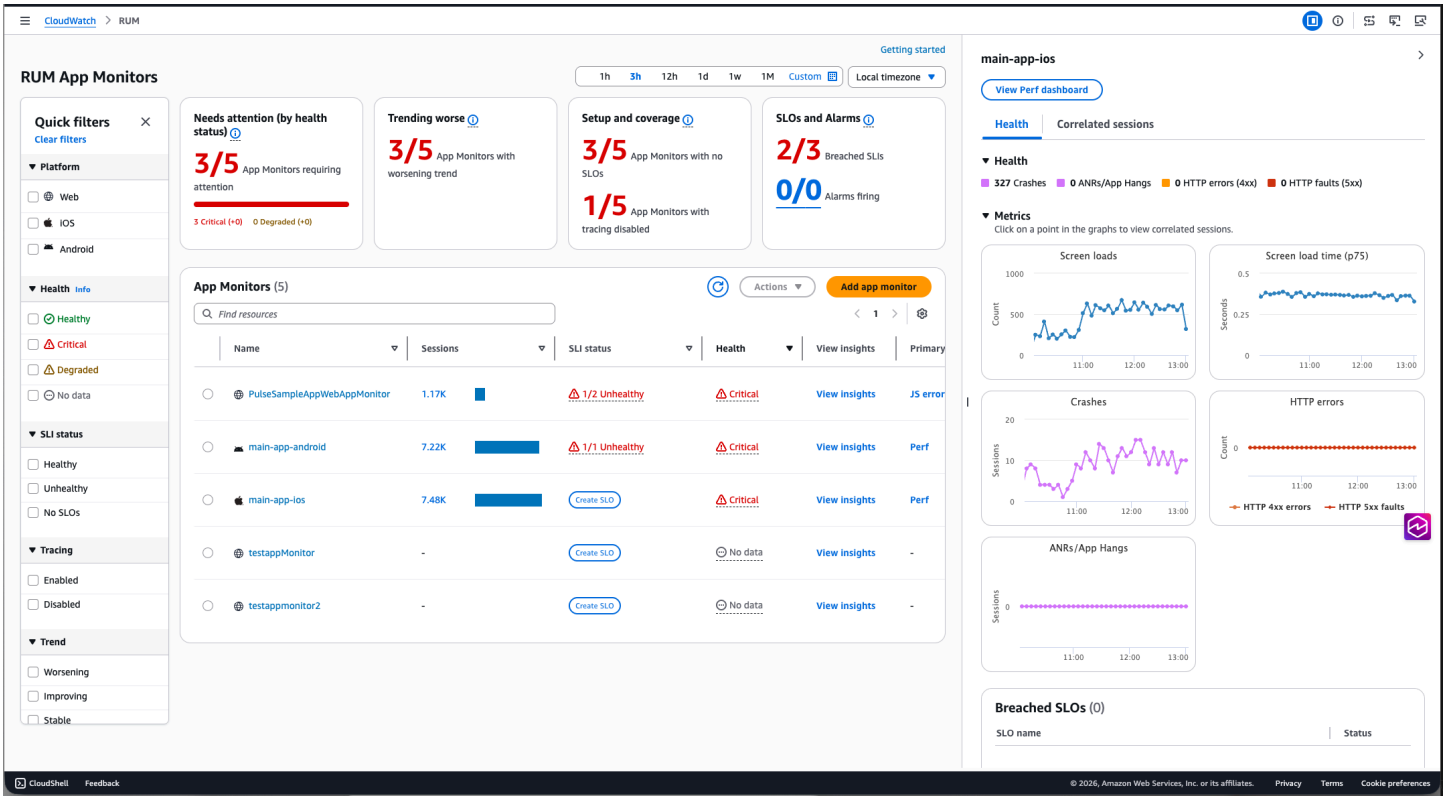
Health tab — Shows a breakdown of errors contributing to the health status (for example, JS errors, HTTP errors (4xx), and HTTP faults (5xx)) with color-coded indicators. The **Metrics** section displays interactive time-series graphs. Click on a point in the graphs to view correlated sessions.

For **web** app monitors, the following graphs are displayed:

- **Page views** — The count of page views over time.
- **Page load time (p75)** — The 75th percentile page load time in seconds.
- **JS errors** — The count of JavaScript error sessions over time.
- **HTTP errors** — The count of HTTP 4xx errors and 5xx faults over time.

For **mobile** app monitors (Android and iOS), the following graphs are displayed:

- **Screen load time** — The screen load time over time.
- **Screen loads** — The count of screen loads over time.
- **Crashes** — The count of crash sessions over time.
- **App Hangs/ANRs** — The count of App Hang (iOS) or ANR (Android) sessions over time.
- **HTTP errors** — The count of HTTP 4xx errors and 5xx faults over time.



Below the metrics, the panel also shows:

- **Breached SLOs** — A table listing any SLOs in a **Breaching** state, with links to the SLO details.
- **Alarms in ALARM state** — A table listing any alarms currently firing for the app monitor.

Correlated sessions tab — Shows sessions correlated to the selected data point in the metrics graphs.

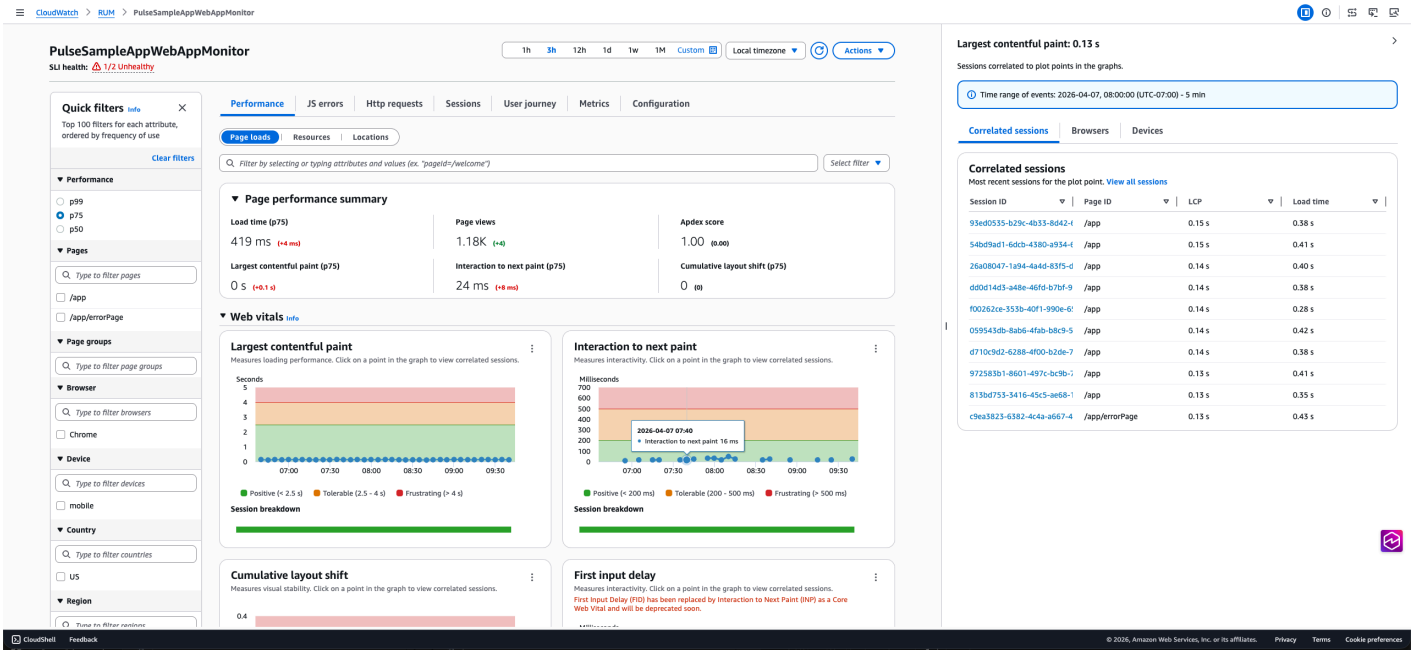
Web Application Dashboard

When you select a web application monitor, you'll see the following tabs:

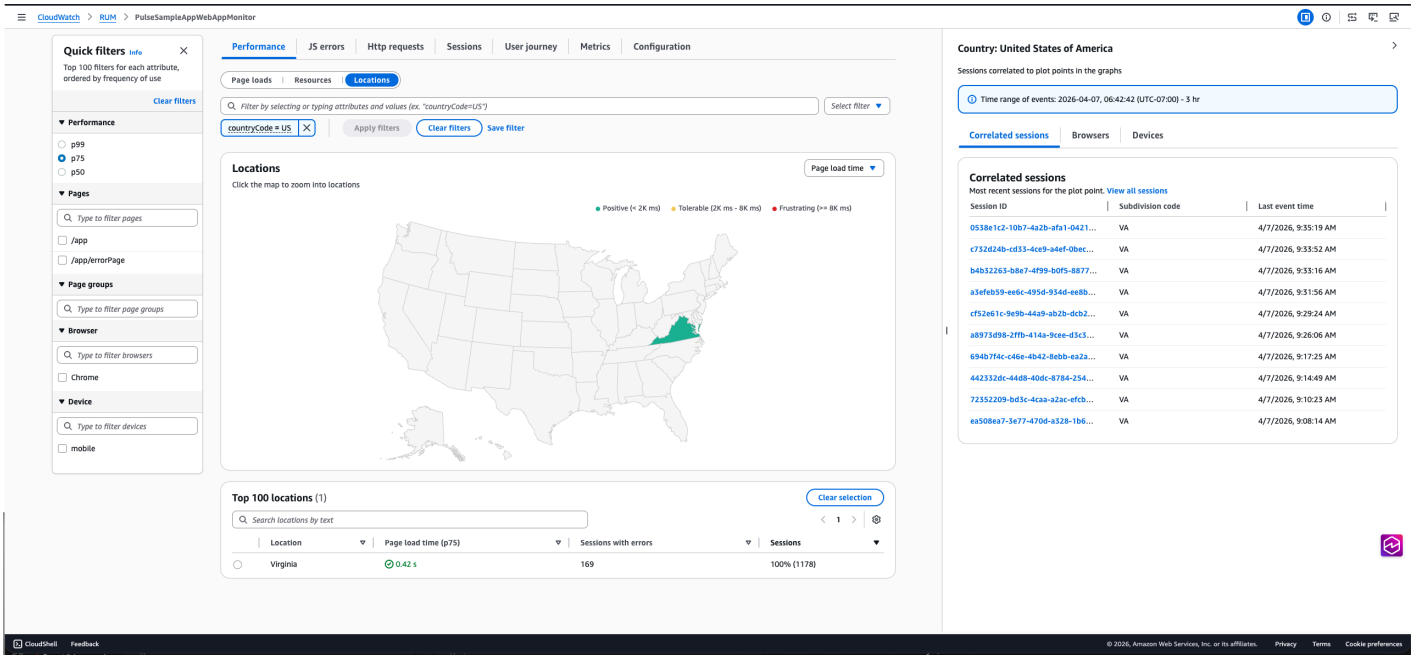
- The **Performance** tab displays page performance information including load times, request information, web vitals, and page loads over time. On this tab you can also toggle the view between **Page loads**, **Resources**, and **Locations** to see more details about page performance.

  The **Page loads** view features interactive web vitals graphs where you can see the different percentile values of core web vitals for your pages and choose datapoints on the graph to view correlated sessions captured by CloudWatch RUM. From there you can navigate to the Sessions tab using one of the links in the diagnostic table to identify specific conditions causing

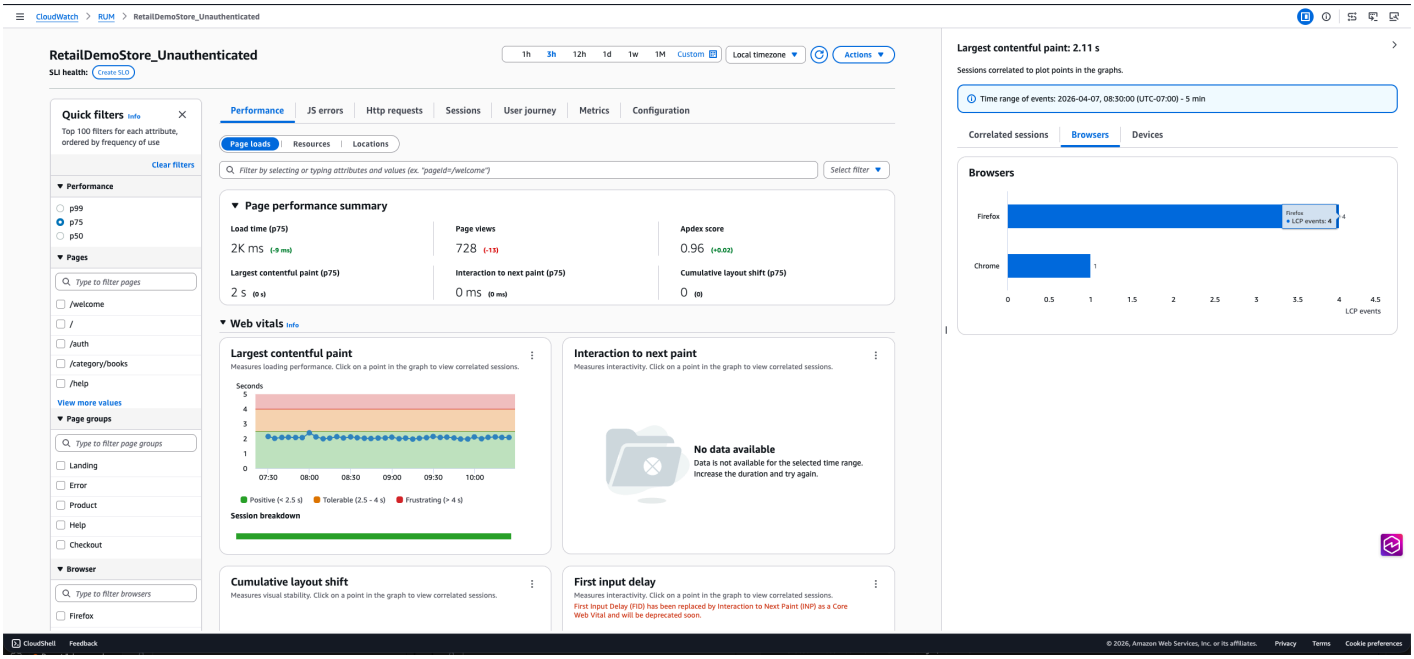
performance issues. The tab also features the application performance index (**Apdex**) score which indicates end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. For more information about Apdex scores, see How CloudWatch RUM sets Apdex scores. The table at the bottom lists the **Top 100 page load times** based on **Page Ids**. You can change this attribute in the dropdown next to the table header.



Similarly the **Resources** view shows the resource request time and count by the resource type. The **Locations** view has interactive map that lets you drill down to a more granular view and investigate performance issues in a specific region.

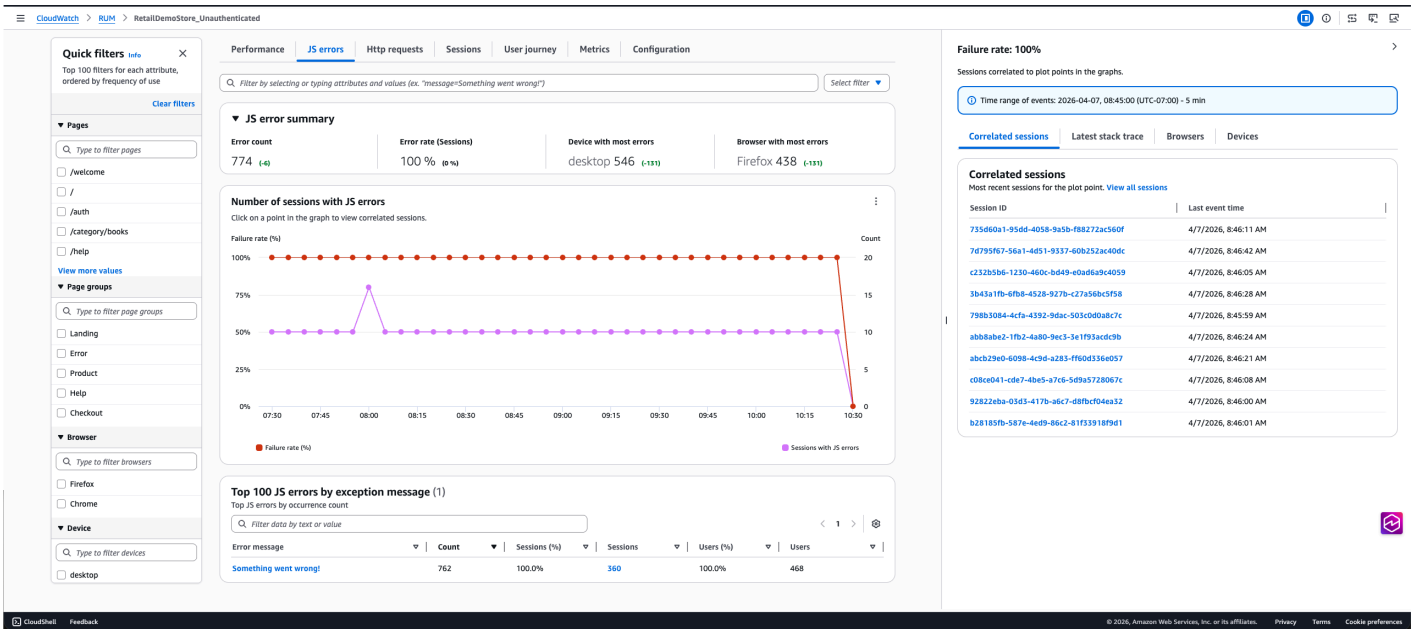


The diagnostic panel on the right also has the **Browsers** and **Devices** tab which shows the top 5 browsers/devices contributing to the performance issue. You can click on the bar chart to navigate to the Sessions tab to further investigate the issue.

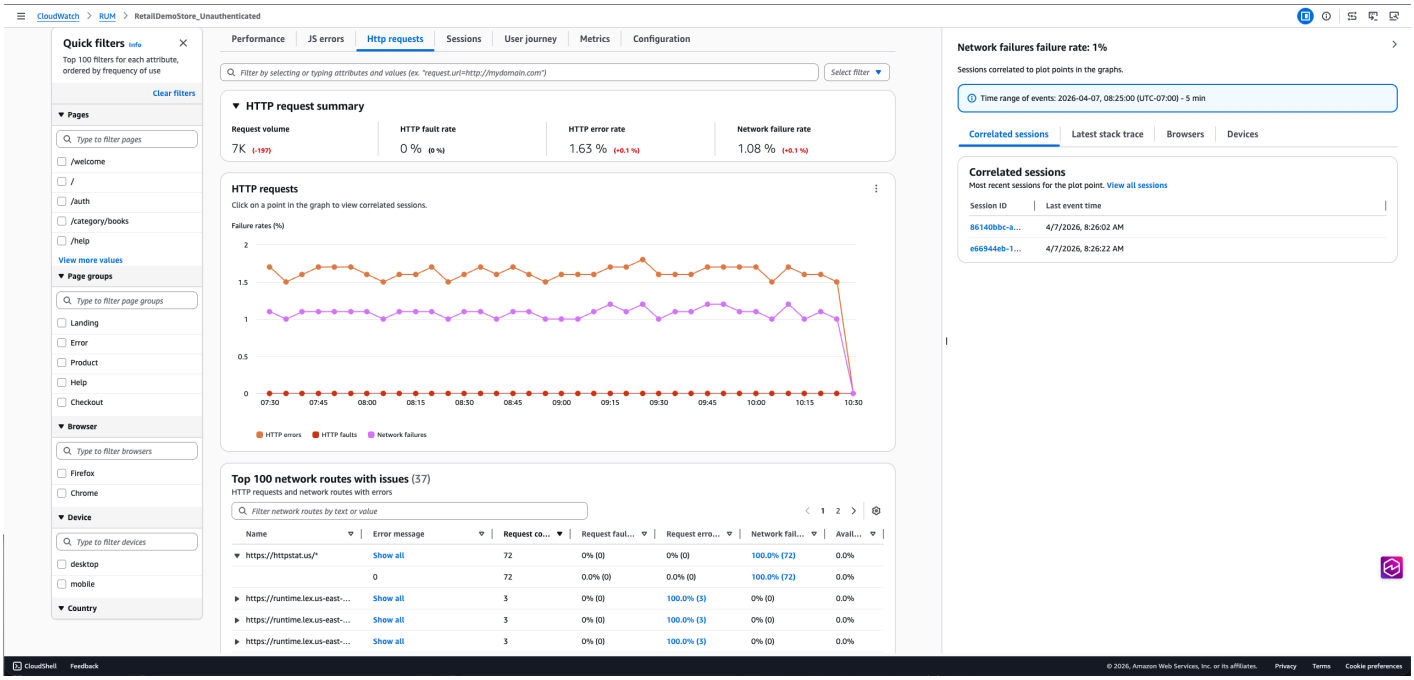


- The **JS errors** tab displays JavaScript error count and rate in the summary component along with the Browser and Device with the most errors. This tab includes a chart which shows the number of sessions with JS errors and the failure rate. You can click on any data point in the chart to view the correlated sessions in the diagnostic panel. The table at the bottom lists the top 100 JS

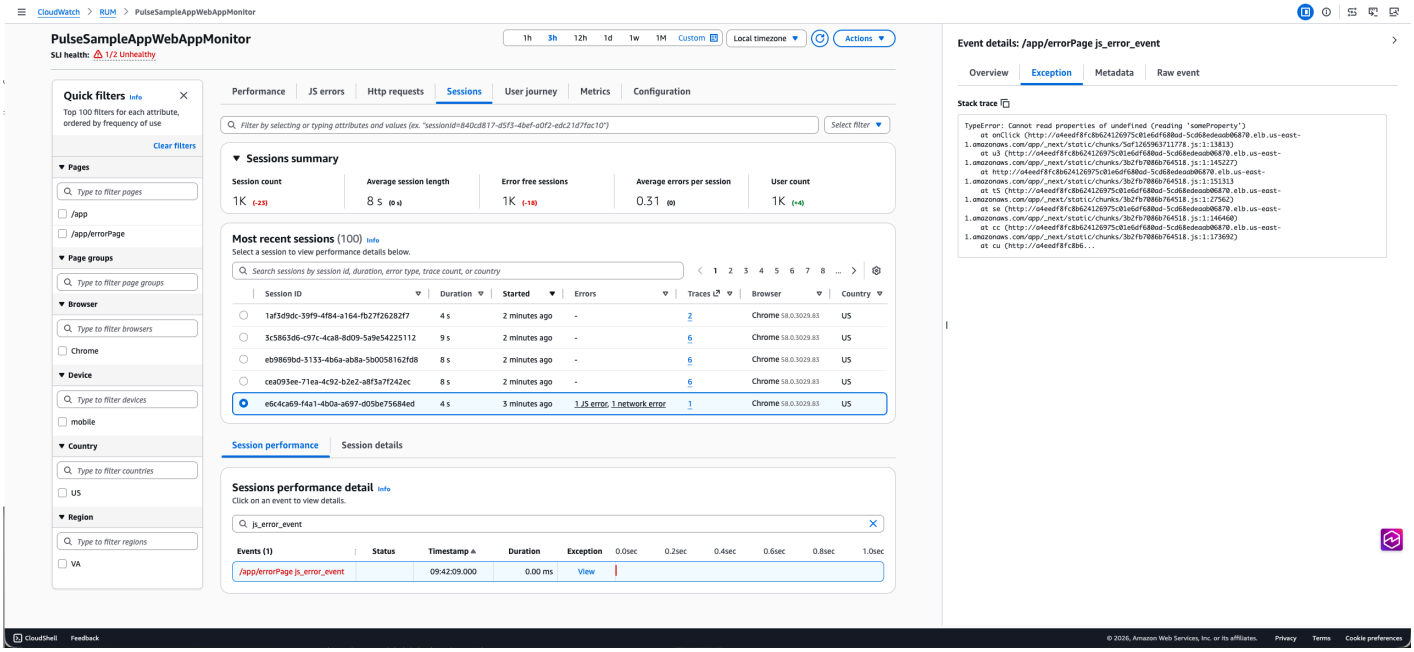
errors. The error count link in the table can be used to navigate to the sessions tab where you can view relevant sessions.



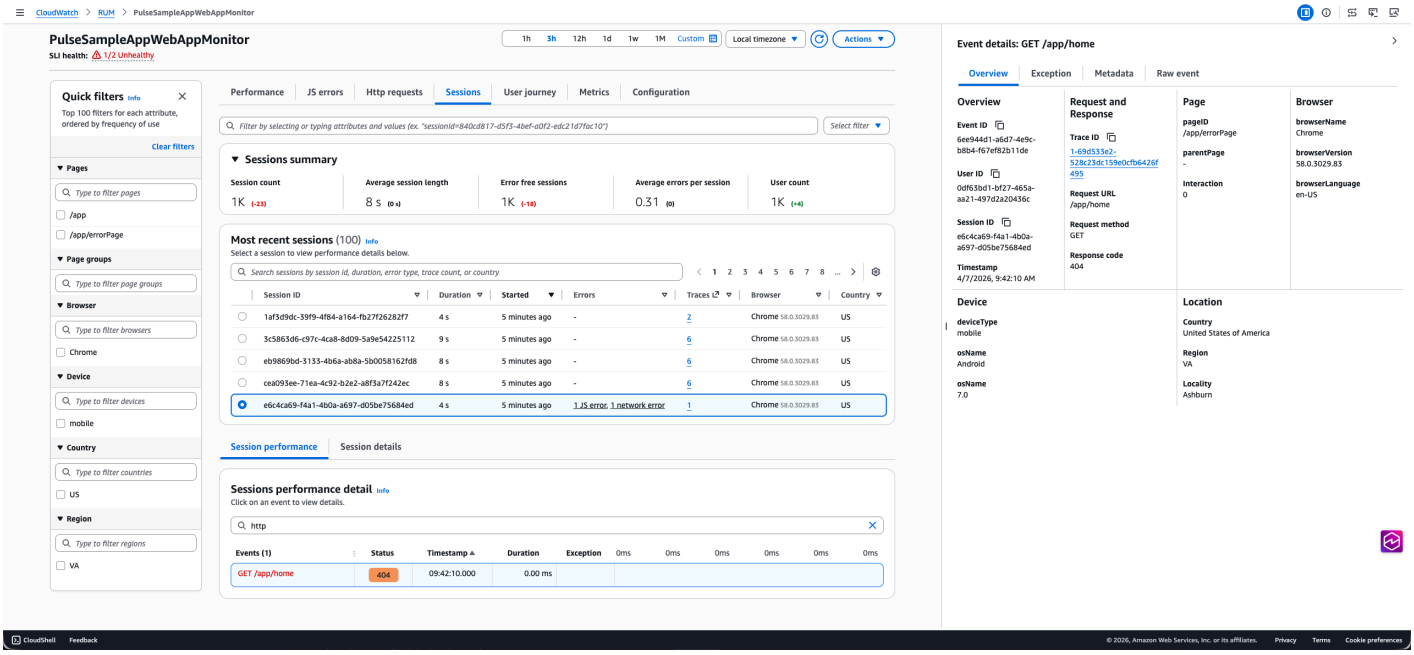
- The **Http requests** tab displays HTTP request volume and error information in the HTTP request summary on the top. This tab includes a graph with the HTTP errors, HTTP faults and Network failures. You can click on any data point in the chart to view the correlated sessions in the diagnostic panel. The table at the bottom lists the top 100 network routes with issues. If you expand one of the rows you can see the top error messages for that url. The error count link in the table can be used to navigate to the sessions tab where you can view relevant sessions.



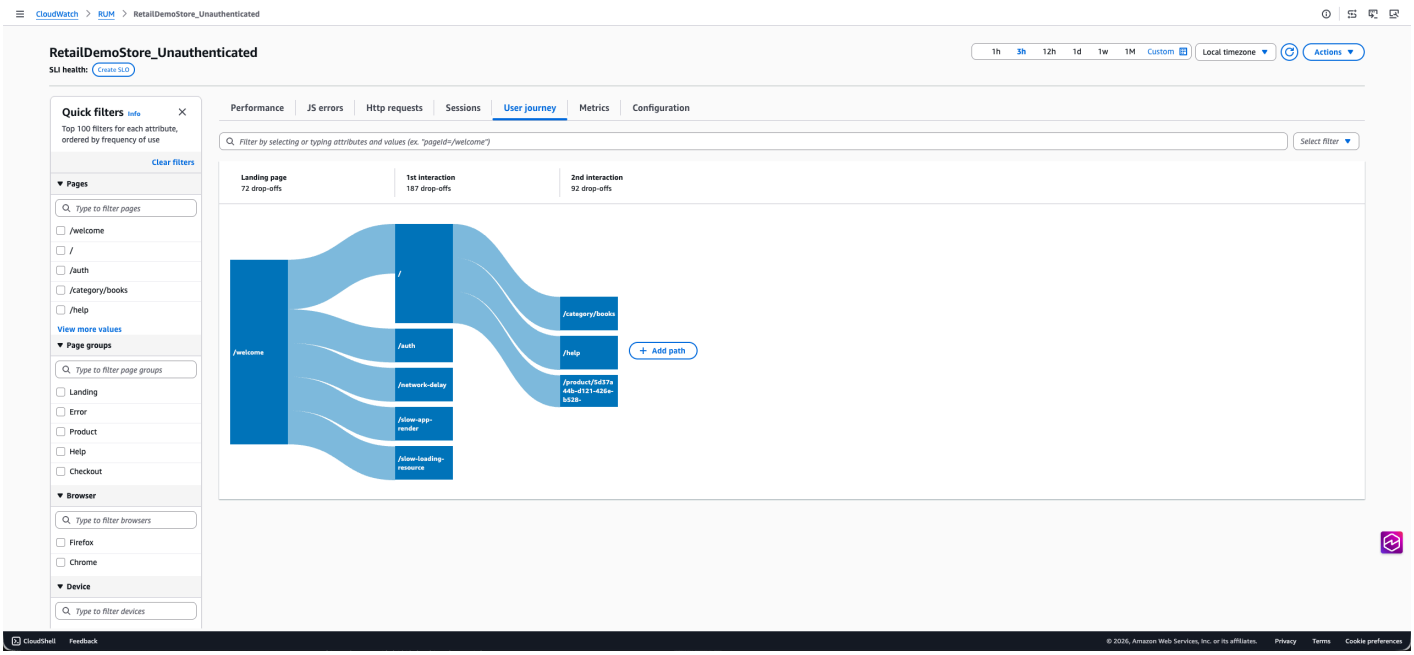
- The **Sessions** tab displays a table that lists all sessions in descending chronological order. At the bottom, a waterfall visualization shows all telemetry for the selected session, helping you track user interactions and identify performance issues. You can click on the error link in the **Errors** column to filter the waterfall chart for the specific error event. Each row in the waterfall can be selected to open the diagnostic panel where you can view the raw event.



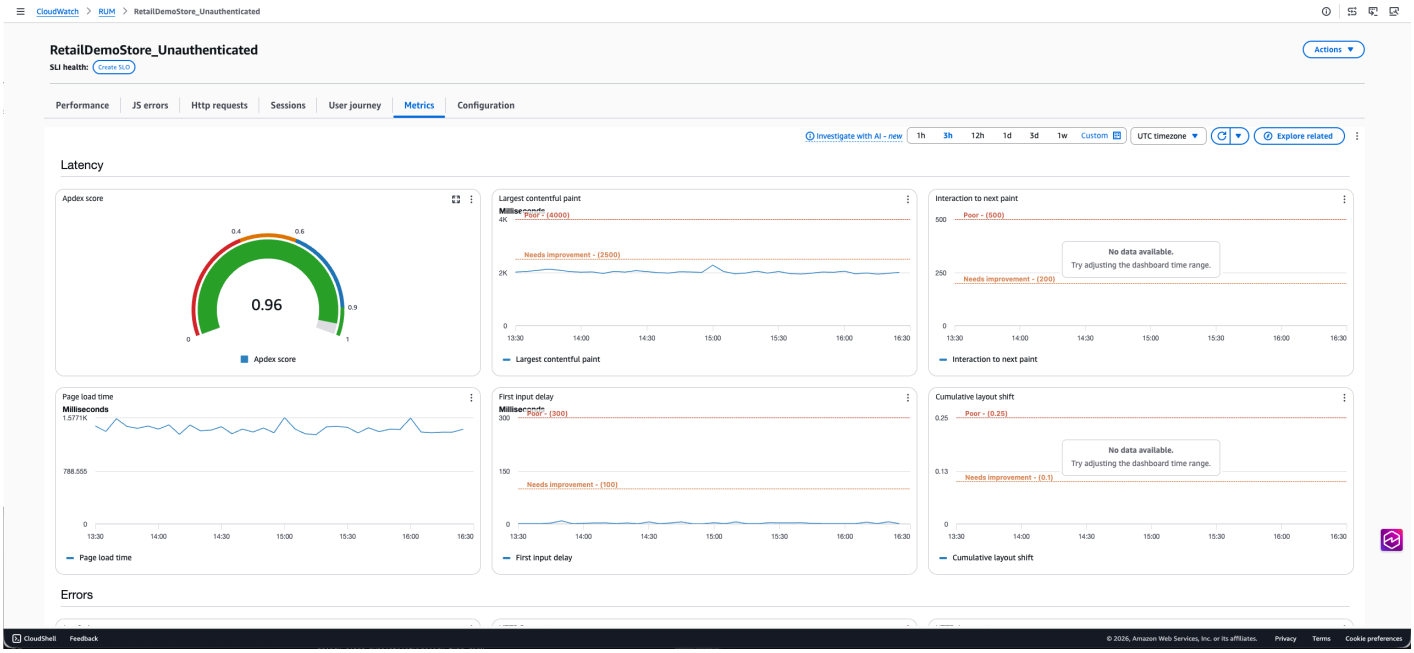
For HTTP requests, you'll see a **traceId** for HTTP and Xray events that links to the Traces console if you have tracing enabled. For events like JS error or HTTP error events, diagnostic panel includes an **Exception** tab with the stack trace. The **View** button in the waterfall provides quick access to this information.



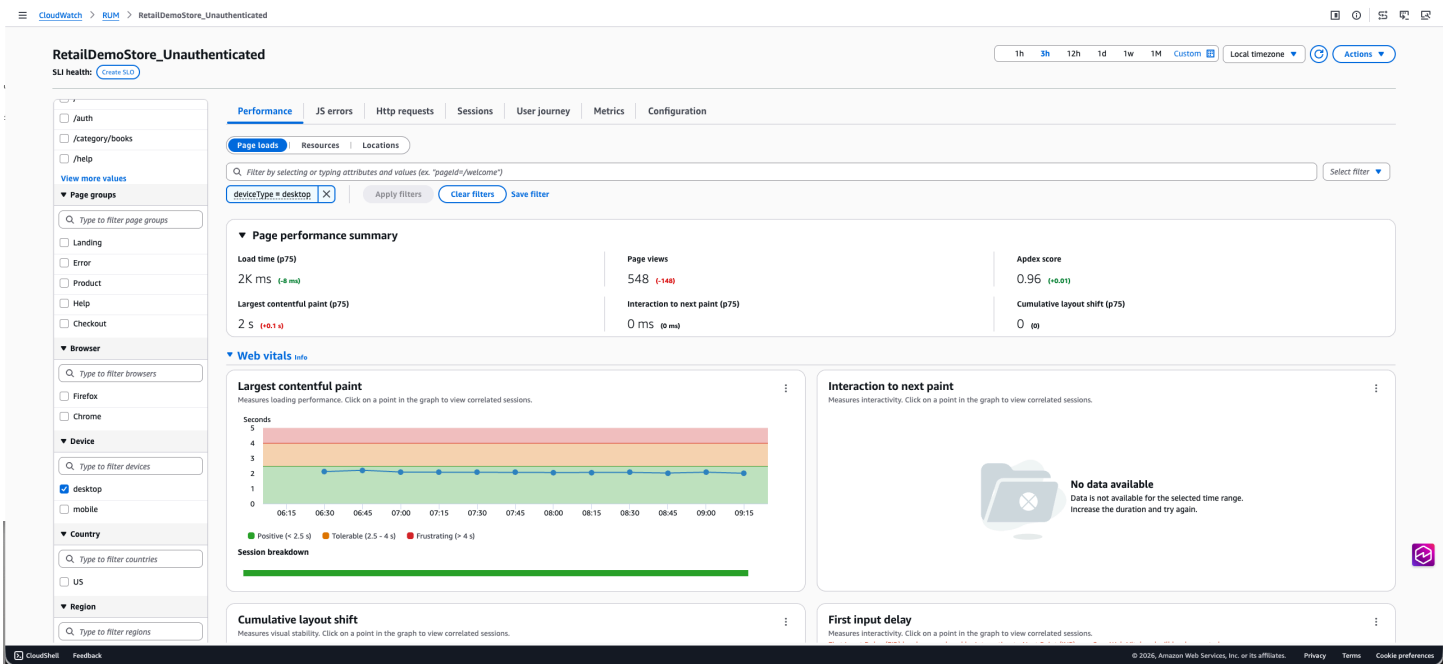
- The **User Journey** tab displays the paths that your customers use to navigate your application. You can see where your customers enter your application and what page they exit your application from. You can also see the paths that they take and the percentage of customers that follow those paths. You can pause on a node to get more details about that page. You can choose a single path to highlight the connections for easier viewing. The page shows user journey till 2nd interaction by default. You can click on the **Add path** button to view further interactions.



- The **Metrics** tab displays all default CloudWatch metrics published by your app monitor, including performance web vitals, error metrics (JavaScript errors, HTTP errors/faults), volume, user flow and apdex metrics. If you created extended metrics for your application, the tab also includes a subset of these metrics in the extended metrics section. This subset includes metrics of type PageViewCount, PerformanceNavigationDuration, Http4xxCount, Http5xxCount and JsErrorCount. The dashboard shows three metric variations per metric type. Since these are CloudWatch metrics, you can also export this tab to your own dashboard using the **Add to dashboard** option and update it to include more metrics.



(Optional) On any of the first five tabs, you can filter the data based on user ID, session ID and other event specific filters using the filter bar on the top. You can also use the quick filter panel on the left to filter on a subset of attributes like Page IDs, Page groups, Device, Browser, Location. These filters can be saved using the **Save filter** option and can be reused using the **Select filter** dropdown next to the filter bar.



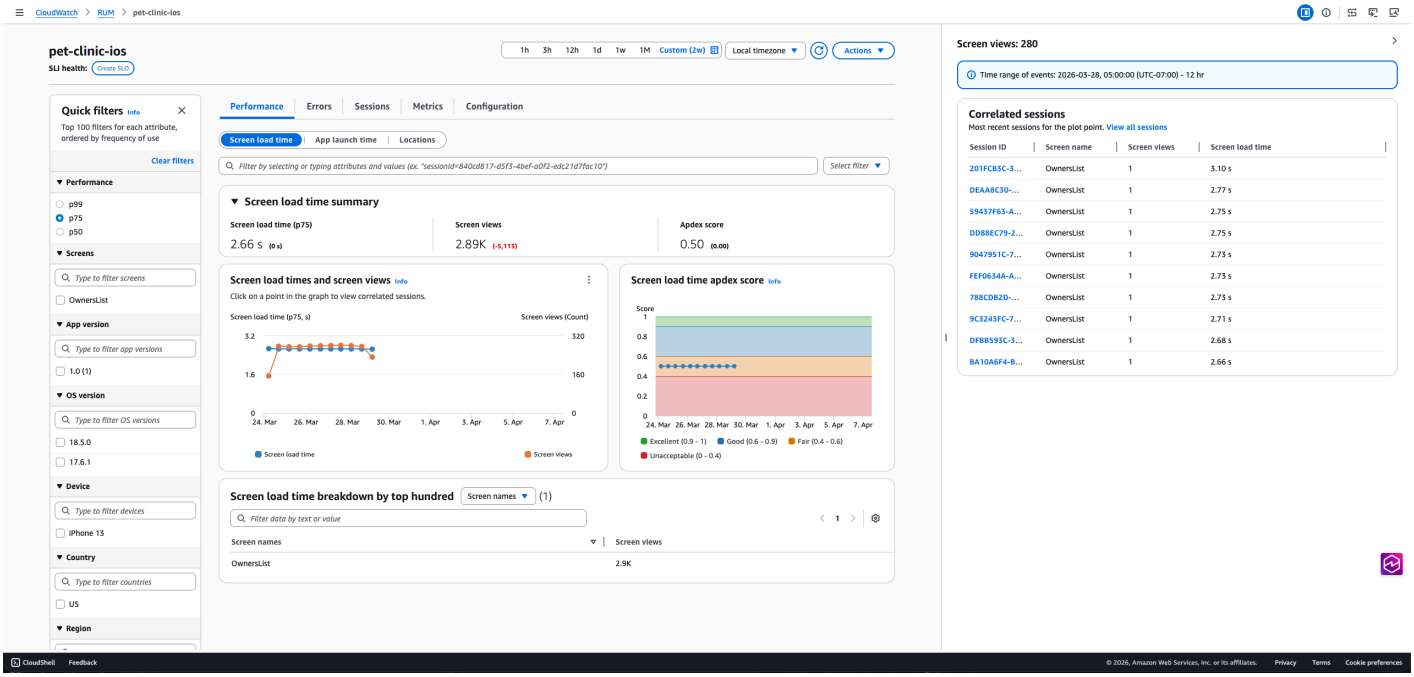
Mobile Application Dashboard

When you select a mobile application monitor, you'll see the following tabs:

- The **Performance** tab provides insights into the performance of your mobile application including screen load times, app launch times (cold and warm), performance metrics, and Apdex scores over time. The detailed view breaks down performance by screen names, OS versions, app versions, devices, and countries. Clicking a screen load time, app launch time, or location datapoint in the chart will open the diagnostic panel on the right that provides further insights relevant to the datapoint consisting the most recent correlated sessions and links to the **Sessions** tab for troubleshooting.

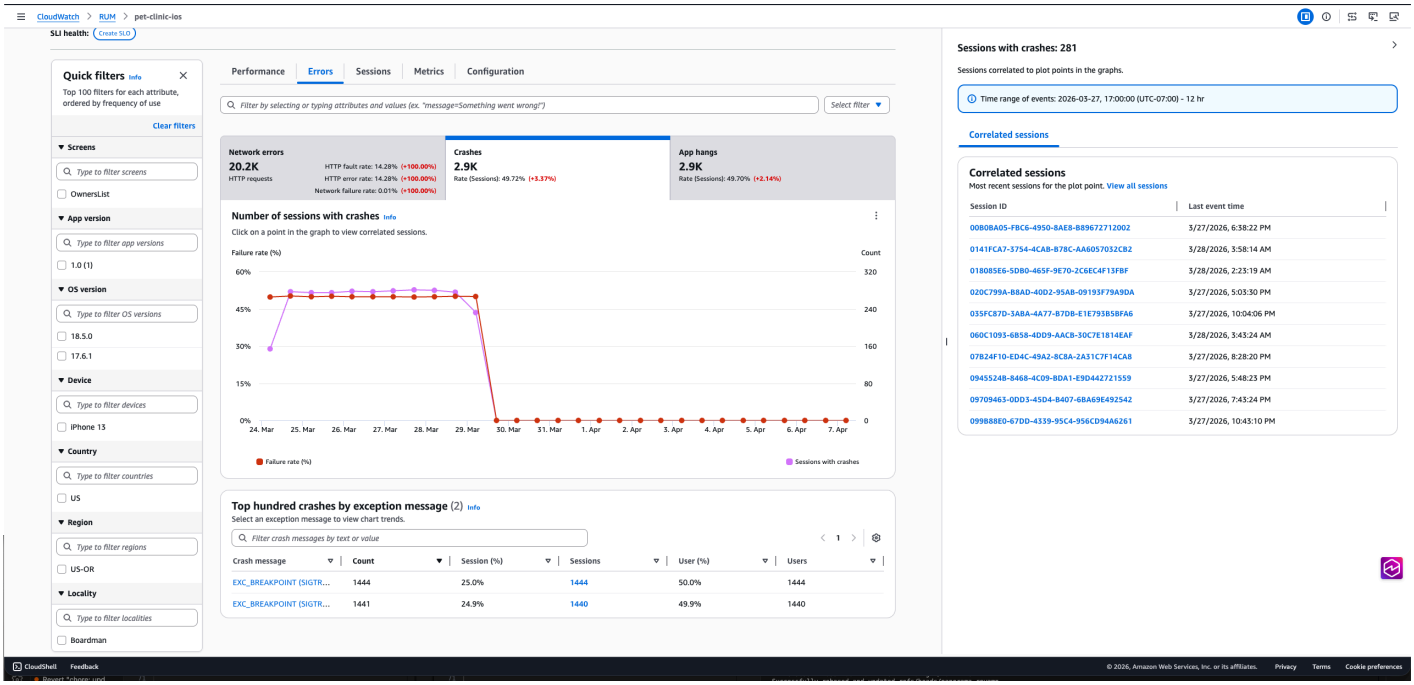
  On this tab you can also toggle the view between **Screen loads**, **App launches**, and **Location** to see more details about application performance.

  The tab also features the application performance index (Apdex) score which indicates end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. For more information about Apdex scores, see How CloudWatch RUM sets Apdex scores.



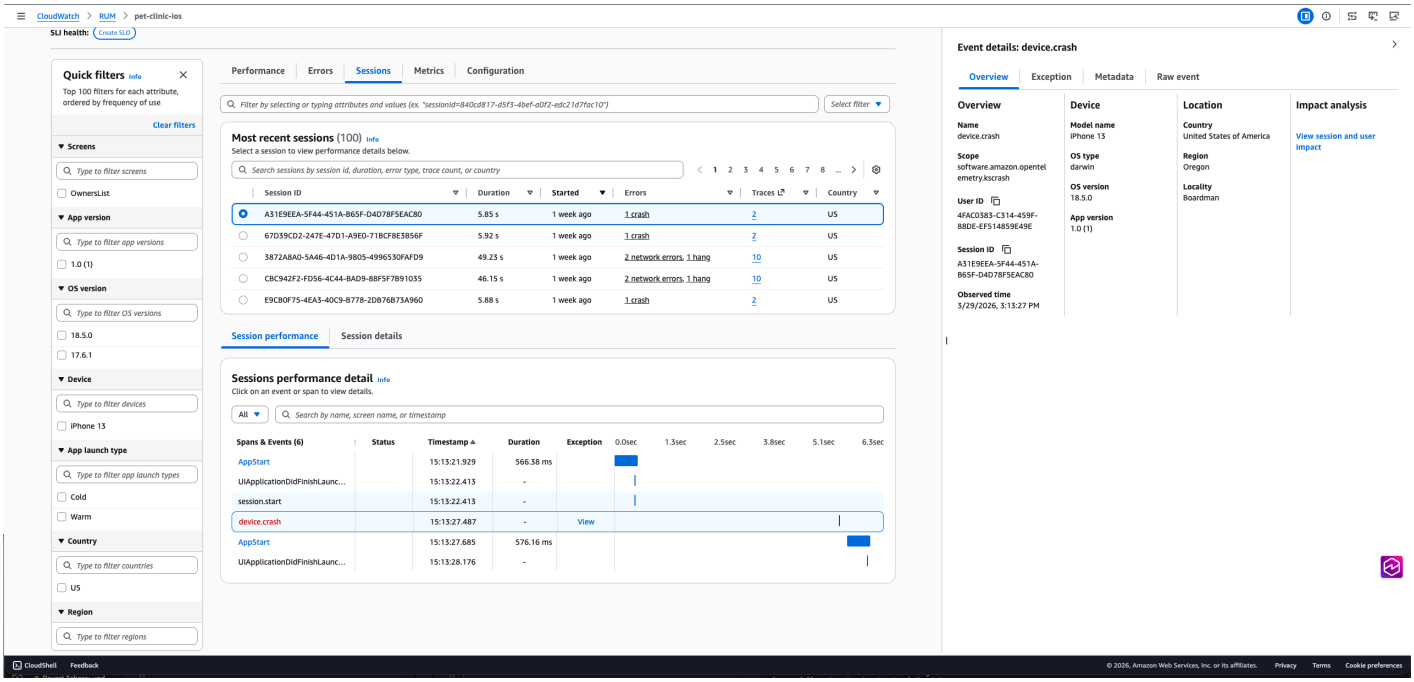
- The **Errors** tab breaks down application issues in three categories: Network Errors, Crashes, and ANRs (Android)/App Hangs (iOS). The **Network Errors** tab has a line chart showing network latency, client errors (4xx status code), and server errors (5xx status code). Clicking on a data point for any of these lines in the chart will open the diagnostic panel. The bottom table lists the 100 most common network routes. Clicking on a radio button will filter the line chart by the network route selected.

  Similarly, the **Crashes** and **ANRs/App Hangs** tabs show a line series for the count of each error, and these are intractable. The bottom table displays the most common top crash message or ANR/App Hang stack trace. Clicking on a radio button will filter the chart, and clicking the error message will show the complete stack trace.



- The **Sessions** tab displays a table that lists all sessions in descending chronological order. At the bottom, a waterfall visualization shows all telemetry for the selected session, helping you track user interactions and identify performance issues. Each row in the waterfall can be selected to open the diagnostic panel. For HTTP requests, you'll see a **traceId** that links to the Traces console.

  For HTTP requests with non-2xx status codes, crashes, or ANRs (Android)/ App Hangs (iOS), the diagnostic panel includes an **Exception** tab with the stack trace. The **View** button in the waterfall provides quick access to this information.



- The **Metrics** tab displays all default CloudWatch metrics published by your app monitor, including performance metrics (screen load times, cold app launch times), error metrics (crashes, ANRs/App Hangs, HTTP errors/faults), volume and apdex metrics. If you created extended metrics for your application, the tab also includes a subset of these metrics in the extended metrics section. This subset includes metrics of type ScreenLoadTime, ScreenLoadCount, CrashCount, Http4xxCount, Http5xxCount, ANRCount/AppHangCount, ColdLaunchTime and WarmLaunchTime. The dashboard shows three metric variations per metric type. Since these are CloudWatch metrics, you can also export this tab to your own dashboard using the **Add to dashboard** option and update it to include more metrics.
- The **Configuration** tab provides access to your app monitor's general settings and configuration details. You can also access the **Code snippets** tab which contains instructions for instrumenting your mobile application with the ADOT SDK, including both Manual and Zero-Code instrumentation options.

How CloudWatch RUM sets Apdex scores

Apdex (Application Performance Index) is an open standard that defines a method to report, benchmark, and rate application response time. An Apdex score helps you understand and identify the impact on application performance over time.

The Apdex score indicates the end users' level of satisfaction. Scores range from 0 (least satisfied) to 1 (most satisfied). The scores are based on application performance only. Users are not asked to rate the application.

Each individual Apdex score falls into one of three thresholds. Based on the Apdex threshold and actual application response time, there are three kinds of performance, as follows:

- **Satisfied**— The actual application response time is less than or equal to the Apdex threshold. For CloudWatch RUM, this threshold is 2000 ms or less.
- **Tolerable**— The actual application response time is greater than the Apdex threshold, but less than or equal to four times the Apdex threshold. For CloudWatch RUM, this range is 2000—8000 ms.
- **Frustrating**— The actual application response time is greater than four times the Apdex threshold. For CloudWatch RUM, this range is over 8000 ms.

The total 0-1 Apdex score is calculated using the following formula:

```
(positive scores + tolerable scores/2)/total scores * 100
```

CloudWatch metrics that you can collect with CloudWatch RUM

The tables in this section lists the metrics that you automatically collect with CloudWatch RUM from web applications, mobile applications, or both. You can see these metrics in the CloudWatch console. For more information, see View available metrics.

You can also optionally send extended metrics to CloudWatch. For more information, see Extended metrics.

These metrics are published in the metric namespace named `AWS/RUM`. All of the following metrics are published with an `application_name` dimension. The value of this dimension is the name of the app monitor. Some metrics are also published with additional dimensions, as listed in the following table.

The following table describes the web metrics.

Metric	Unit	Description
`HttpStatusCodeCount`	Count	The count of HTTP responses in the

Metric	Unit	Description
		application, by their response status code. Additional dimensions: • `event_details.response.status` is the response status code, such as 200, 400, 404, and so on. • `event_type` The type of event.
`Http4xxCount`	Count	The count of HTTP responses in the application, with 4xx response status code. These are calculated based on `http_event` RUM events that result in 4xx codes.
`Http4xxCountPerSession`	Count	The count of HTTP responses in a session, with 4xx response status code. These are calculated based on `http_event` RUM events that result in 4xx codes.

Metric	Unit	Description
Http4xxCountPerPageView	Count	The count of HTTP responses in a page review, with 4xx response status code. These are calculated based on http_event RUM events that result in 4xx codes.
Http5xxCount	Count	The count of HTTP responses in the application, with 5xx response status code. These are calculated based on http_event RUM events that result in 5xx codes.
Http5xxCountPerSession	Count	The count of HTTP responses in the session, with 5xx response status code. These are calculated based on http_event RUM events that result in 5xx codes.

Metric	Unit	Description
Http5xxCountPerPageView	Count	The count of HTTP responses in a page review, with 5xx response status code. These are calculated based on http_event RUM events that result in 5xx codes.
JsErrorCount	Count	The count of JavaScript error events ingested.
JsErrorCountPerSession	Count	The count of JavaScript error events ingested in a session.
JsErrorCountPerPageView	Count	The count of JavaScript error events ingested in a page review.
NavigationFrustratedTransaction	Count	The count of navigation events with a duration higher than the frustrating threshold, which is 8000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.

Metric	Unit	Description
NavigationSatisfiedTransaction	Count	The count of navigation events with a duration that is less than the Apdex objective, which is 2000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.
NavigationToleratedTransaction	Count	The count of navigation events with a duration between 2000ms and 8000ms. The duration of navigation events is tracked in the PerformanceNavigationDuration metric.
PageViewCount	Count	The count of page view events ingested by the app monitor. This is calculated by counting the page_view_event RUM events.

Metric	Unit	Description
PageViewCountPerSession	Count	The count of page view events in a session.
PerformanceResourceDuration	Milliseconds	The duration of a resource event. Additional dimensions: • event_details.file.type is the file type of the resource event, such as a stylesheet, document, image, script, or font. • event_type The type of event.
PerformanceNavigationDuration	Milliseconds	The duration of a navigation event.
RumEventPayloadSize	Bytes	The size of every event ingested by CloudWatch RUM. You can also use the SampleCount statistic for this metric to monitor the number of events that an app monitor is ingesting.

Metric	Unit	Description
SessionCount	Count	The count of session start events ingested by the app monitor. In other words, the number of new sessions started.
SessionDuration	Milliseconds	The duration of a session. These are calculated based on the time between first and last events in a session.
TimeOnPage	Milliseconds	The duration of a page view. These are calculate d based on the time until next page view, except for the final page in a session where it's the time between first and last events on that page.
WebVitalsCumulativeLayoutShift	None	Tracks the value of the cumulative layout shift events.
WebVitalsFirstInputDelay	Milliseconds	Tracks the value of the first input delay events.

Metric	Unit	Description
WebVitalsLargestContentfulPaint	Milliseconds	Tracks the value of the largest contentful paint events.
WebVitalsInteractionToNextPaint	Milliseconds	Tracks the value of the interaction to next paint events.

You can configure extended metrics for your mobile application to provide additional dimensions for analysis.

Metric	Unit	Description
ANRCount	Count	For Android only: The number of Application Not Responding (ANR) incidents, occurring when the application is unresponsive for more than 5 seconds, resulting in application crash.
AppHangCount	Count	For iOS only: The number of times the application became unresponsive for more than 250ms on the main loop.
ColdAppLaunchFrustratedTransaction	Count	The number of cold app launches that took longer than 8 seconds to complete, likely causing user frustration.
ColdAppLaunchSatisfiedTransaction	Count	The number of cold app launches that completed in less than 2 seconds, providing a satisfactory user experience.
ColdAppLaunchToleratedTransaction	Count	The number of cold app launches that completed between 2 and 8 seconds, providing a tolerable, but not ideal, user experience.
ColdLaunchTime	Milliseconds	Time taken to launch the application from a terminated state.

Metric	Unit	Description
		For Android: Time from Application onCreate until the first Activity finishes creating. For iOS: Time from application start (determined by sysctl process start command) until didBecome ActiveNotification .
CrashCount	Count	The number of unexpected application terminations caused by unhandled exceptions or OS termination. For Android: Crashes due to unhandled exceptions or system termination. For iOS: Crashes due to unhandled exceptions, fatal errors, or system termination. Crash data is stored locally and reported on next app launch.
DroppedEventsCount	Count	The number of log events that were dropped because they exceeded the maximum size limit of 30KB per event.
DroppedSpansCount	Count	The number of spans that were dropped because they exceeded the maximum size limit of 30KB per span.
Http4xxCount	Count	Records the number of HTTP client errors encounter ed by the web or mobile application during HTTP requests.
Http5xxCount	Count	Records the number of HTTP server errors encounter ed by the web or mobile application during HTTP requests.

Metric	Unit	Description
LogPayloadSize	Bytes	The size in bytes of the log telemetry data being sent to CloudWatch RUM. You can also use the SampleCount statistic for this metric to monitor the number of Log Events that an app monitor is ingesting.
NetworkLatency	Milliseconds	The time taken for network requests to complete, measuring the round-trip time from request initiation to response completion.
ScreenLoadCount	Count	The total number of screen loads.
ScreenLoadToleratedTransaction	Count	The number of screen loads that completed between 2 and 8 seconds, providing a tolerable, but not ideal, user experience.
SessionCount	Count	The total number of unique user sessions with the application. A session begins when the user opens the app and ends after 30 minutes of inactivity or when explicitly terminated.
SpanPayloadSize	Bytes	The size in bytes of the span telemetry data being sent to CloudWatch RUM. You can also use the SampleCount statistic for this metric to monitor the number of Spans that an app monitor is ingesting.
WarmAppLaunchFrustratedTransaction	Count	The number of warm app launches that took longer than 8 seconds to complete, likely causing user frustration.
WarmAppLaunchSatisfiedTransaction	Count	The number of warm app launches that completed in less than 2 seconds, providing a satisfactory user experience.

Metric	Unit	Description
`WarmAppLaunchTolerated Transaction`	Count	The number of warm app launches that completed between 2 and 8 seconds, providing a tolerable, but not ideal, user experience.
`WarmLaunchTime`	Milliseconds	Time taken to launch the application from background state. For Android: Time from Application on`Create` until the first Activity finishes creating. For iOS: Time from `UIApplicationWillEnterForegroundNotification` until `didBecomeActiveNotification` .

Custom metrics and extended metrics that you can send to CloudWatch

By default, RUM app monitors send metrics to CloudWatch. These default metrics and dimensions are listed in CloudWatch metrics that you can collect with CloudWatch RUM.

You can also set up an app monitor to export metric. The app monitor can send extended metrics, custom metrics, or both. It can send them to CloudWatch.

- **Custom metrics**— Custom metrics are metrics that you define. With custom metrics, you can use any metric name and namespace. To derive the metrics, you can use any custom events, built-in events, custom attributes, or default attributes.

  You can send custom metrics to CloudWatch.
- **Extended metrics**— You can send any of the default CloudWatch RUM metrics to CloudWatch with additional dimensions. This way, these metrics can give you a more fine-grained view.

Topics

- Custom metrics
- Extended metrics

Custom metrics

To send custom metrics, you must use the AWS APIs or AWS CLI instead of the console. For more information about using the AWS APIs, see PutRumMetricsDestination and BatchCreateRumMetricDefinitions.

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each custom metric or extended metric that you send to each destination, each combination of dimension name and dimension value counts toward this limit. You are not charged for custom metrics derived from any kind of events or attributes of CloudWatch RUM.

The following example shows how to create a custom metric derived from a custom event. Here is the example custom event that is used:

```
cwr('recordEvent', {
    type: 'my_custom_event',
    data: {
        location: 'IAD',
        current_url: 'amazonaws.com',
        user_interaction: {
            interaction_1 : "click",
            interaction_2 : "scroll"
        },
        visit_count:10
    }
})
```

Given this custom event, you can create a custom metric that counts the number of visits to the `amazonaws.com` URL from Chrome browsers. The following definition creates a metric named `AmazonVisitsCount` in your account, in the `RUM/CustomMetrics/PageVisits` namespace.

```
{
    "AppMonitorName":"customer-appMonitor-name",
    "Destination":"CloudWatch",
    "MetricDefinitions":[
        {
            "Name":"AmazonVisitsCount",
            "Namespace":"PageVisit",
            "ValueKey":"event_details.visit_count",
            "UnitLabel":"Count",
            "DimensionKeys":{
                "event_details.current_url": "URL"
```

```
            },
            "EventPattern":"{\"metadata\":{\"browserName\":[\"Chrome\"]},\"event_type
\":[\"my_custom_event\"],\"event_details\": {\"current_url\": [\"amazonaws.com\"]}}"
        }
    ]
}
```

Extended metrics

If you set up extended metrics, you can send any of the default CloudWatch RUM metrics to CloudWatch with additional dimensions so that the metrics give you a more fine-grained view.

For more information about the default CloudWatch RUM metrics, see CloudWatch metrics that you can collect with CloudWatch RUM.

The maximum number of extended metric and custom metric definitions that one destination can contain is 2000. For each extended or custom metric that you send to each destination, each combination of dimension name and dimension value counts as an extended metric for this limit.

When you send extended metrics to CloudWatch, you can use the CloudWatch RUM console to create CloudWatch alarms on them.

You are not charged for extended metrics that are created for the default metrics of CloudWatch RUM.

Web Application Extended Metrics

The following dimensions are supported for web application extended metrics:

- `BrowserName`

  Example dimension values: `Chrome`, `Firefox`, `Chrome Headless`
- `CountryCode` This uses the ISO-3166 format, with two-letter codes.

  Example dimension values: `US`, `JP`, `DE`
- `DeviceType`

  Example dimension values: `desktop`, `mobile`, `tablet`, `embedded`
- `FileType`

  Example dimension values: `Image`, `Stylesheet`

- `OSName`

  Example dimension values: `Linux`, `Windows`, `iOS`, `Android`
- `PageId`

Mobile Application Extended Metrics

The following dimensions are supported for mobile application extended metrics:

- `ScreenName`
  - Value from the screen name attribute - `attributes.screen.name` of your application
  - Example values: HomeScreen, SettingsView, ProfilePage
- `DeviceModel`
  - Value from the device model attribute - `resource.attributes.device.model.name`
  - Example values: iPhone14,3, SM-G998B
- `OSVersion`
  - Value from the operating system version attribute - `resource.attributes.os.version`
  - Example values: 16.0, 13.0

> **Note**
>
> Currently supported default metrics for mobile applications: `ScreenLoadTime`, `ScreenLoadCount`, `NetworkLatency`, `Http4xxCount`, `Http5xxCount`, `CrashCount`, `ANRCount`, `AppHangCount`, `ColdLaunchTime`, `WarmLaunchTime`, `SessionCount`, `SpanPayloadSize` and `LogPayloadSize`.

Set up extended metrics using the console

To use the console to send extended metrics to CloudWatch, use the following steps.

For information about using the AWS APIs to send extended metrics to either CloudWatch, see PutRumMetricsDestination and BatchCreateRumMetricDefinitions.

To use the console to set up an app monitor and send RUM extended metrics to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Choose **Send metrics**.
6. Select one or more metric names to send with additional dimensions.
7. Select one or more factors to use as dimensions for these metrics. As you make your choices, the number of extended metrics that your choices create is displayed in **Number of extended metrics**.

   This number is calculated by multiplying the number of chosen metric names by the number of different dimensions that you create.

   a. To send a metric with page ID as a dimension, choose **Browse for page ID** and then select the page IDs to use.
   b. To send a metric with device type as a dimension, choose either **Desktop devices** or **Mobile and tablets**.
   c. To send a metric with operating system as a dimension, select one or more operating systems under **Operating system**.
   d. To send a metric with browser type as a dimension, select one or more browsers under **Browsers**.
   e. To send a metric with geolocation as a dimension, select one or more locations under **Locations**.

      Only the locations where this app monitor has reported metrics from will appear in the list to choose from.
8. When you are finished with your choices, choose **Send metrics**.
9. (Optional) In the **Extended metrics** list, to create an alarm that watches one of the metrics, choose **Create alarm** in that metric's row.

   For general information about CloudWatch alarms, see Using Amazon CloudWatch alarms. For a tutorial for setting an alarm on a CloudWatch RUM extended metric, see Tutorial: create an extended metric and alarm it.

Stop sending extended metrics

To use the console to stop sending extended metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose the name of the app monitor that is to send the metrics.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Select one or more metric name and dimension combinations to stop sending. Then choose **Actions**, **Delete**.

Tutorial: create an extended metric and alarm it

This tutorial demonstrates how to set up an extended metric to be sent to CloudWatch, and then how to set an alarm on that metric. In this tutorial, you create a metric that tracks JavaScript errors on the Chrome browser.

To set up this extended metric and set an alarm on it

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose the name of the app monitor that is to send the metric.
4. Choose the **Configuration** tab and then choose **RUM extended metrics**.
5. Choose **Send metrics**.
6. Select **JSErrorCount**.
7. Under **Browsers**, select **Chrome**.

   This combination of **JSErrorCount** and **Chrome** will send one extended metric to CloudWatch. The metric counts JavaScript errors only for user sessions that use the Chrome browser. The metric name will be **JsErrorCount** and the dimension name will be **Browser**.
8. Choose **Send metrics**.
9. In the **Extended metrics** list, choose **Create alarm** in the row that displays **JsErrorCount** under **Name** and displays **Chrome** under **BrowserName**.
10. Under **Specify metric and conditions**, confirm that the **Metric name** and **BrowserName** fields are pre-filled with the correct values.
11. For **Statistic**, select the statistic that you want to use for the alarm. **Average** is a good choice for this type of counting metric.

12. For **Period**, select **5 minutes**.
13. Under **Conditions**, do the following:
    - Choose **Static**.
    - Choose **Greater** to specify that the alarm should go into ALARM state when the number of errors is higher than the threshold you are about to specify.
    - Under **than...**, enter the number for the alarm threshold. The alarm goes into ALARM state when the number of errors over a 5-minute period exceeds this number.
14. (Optional) By default, the alarm goes into ALARM state as soon as the number of errors exceeds the threshold number you set during a 5-minute period. You can optionally change this so that the alarm goes into ALARM state only if this number is exceeded for more than one 5-minute period.

    To do so, choose **Additional configuration** and then for **Datapoints to alarm**, specify how many 5-minute periods need to have the error number over the threshold to trigger the alarm. For example, you can select 2 out of 2 to have the alarm trigger only when two consecutive 5-minute periods are over the threshold, or 2 out of 3 to have the alarm trigger if any two of three consecutive 5-minute periods are over the threshold.

    For more information about this type of alarm evaluation, see Alarm evaluation.
15. Choose **Next**.
16. For **Configure actions**, specify what should happen when the alarm goes into alarm state. To receive a notification with Amazon SNS, do the following:
    - Choose **Add notification**.
    - Choose **In alarm**.
    - Either select an existing SNS topic or create a new one. If you create a new one, specify a name for it and add at least one email address to it.
17. Choose **Next**.
18. Enter a name and optional description for the alarm, and choose **Next**.
19. Review the details and choose **Create alarm**.

Data protection and data privacy with CloudWatch RUM

The AWS shared responsibility model applies to data protection and data privacy in Amazon CloudWatch RUM. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see The AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog. For more resources about complying with GDPR requirements, see the General Data Protection Regulation (GDPR) Center.

Amazon CloudWatch RUM generates a code snippet for you to embed into your website or web application code, based on your input of end user data that you want to collect. The web client, downloaded and configured by the code snippet, uses cookies (or similar technologies) to help you collect end user data. The use of cookies (or similar technologies) is subject to data privacy regulations in certain jurisdictions. Before using Amazon CloudWatch RUM, we strongly recommend that you assess your compliance obligations under applicable law, including any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for the use of cookies and the processing (including collection) of end user data. For more information about how the web client uses cookies (or similar technologies) and what end-user data the web client collects, see Information collected by the CloudWatch RUM web client and CloudWatch RUM web client cookies (or similar technologies).

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Amazon CloudWatch RUM or other services might be included in diagnostic logs.

CloudWatch RUM web client cookies (or similar technologies)

The CloudWatch RUM web client collects certain data about user sessions by default. You can choose to enable cookies to have the web client collect a user ID and session ID that persist across page loads. The user ID is randomly generated by RUM.

If these cookies are enabled, RUM is able to display the following types of data when you view the RUM dashboard for this app monitor.

- Aggregated data based on user IDs, such as number of unique users and the number of different users who experienced an error.

- Aggregated data based on session IDs, such as number of sessions and the number of sessions that experienced an error.
- The *user journey*, which is the sequence of pages that each sampled user session includes.

> **Important**
>
> If you do not enable these cookies (or similar technologies), the web client still records certain information about end user sessions such as browser type/version, operating system type/version, device type, and so on. These are collected to provide aggregated page-specific insights, such as web vitals, page views, and pages that experienced errors. For more information about the data recorded, see Information collected by the CloudWatch RUM web client.

Information collected by the CloudWatch RUM web client

This section documents the **PutRumEvents** schema, which defines the structure of the data that you can collect from user sessions using CloudWatch RUM.

A **PutRumEvents** request sends a data structure with the following fields to CloudWatch RUM.

- The ID of this batch of RUM events
- App monitor details, which includes the following:
  - App monitor ID
  - Monitored application version
- User details, which includes the following. **This is collected only if the app monitor has cookies enabled.**
  - A user ID generated by the web client
  - Session ID
- The array of RUM events in this batch.

RUM event schema

The structure of each RUM event includes the following fields.

- The ID of the event

- A timestamp
- The event type
- The user agent
- Metadata
- RUM event details

RUM event metadata

The metadata includes page metadata, user agent metadata, geolocation metadata, and domain metadata.

Page metadata

The page metadata includes the following:

- Page ID
- Page title
- Parent page ID. – **This is collected only if the app monitor has cookies enabled.**
- Interaction depth – **This is collected only if the app monitor has cookies enabled.**
- Page tags – You can add tags to page events to group pages together. For more information, see Use page groups.

User agent metadata

The user agent metadata includes the following:

- Browser language
- Browser name
- Browser version
- Operating system name
- Operating system version
- Device type
- Platform type

Geolocation metadata

The geolocation metadata includes the following:

- Country code
- Subdivision code

Domain metadata

The domain metadata includes the URL domain.

RUM event details

The details of an event follow one of the following type of schemas, depending on the event type.

Session start event

This event contains no fields. **This is collected only if the app monitor has cookies enabled.**

Page view schema

A **Page view** event contains the following properties. You can deactivate page view collection by configuring the web client. For more information, see the CloudWatch RUM web client documentation.

Name	Type	Description
Page ID	String	An ID that uniquely represents this page within the application. By default, this is the URL path.
Parent page ID	String	The ID of the page that the user was on when they navigated to the current page. **This is collected only if the app monitor has cookies enabled.**
Interaction depth	String	**This is collected only if the app monitor has cookies enabled.**

JavaScript error schema

JavaScript error events generated by the agent contain the following properties. The web client collects these events only if you selected to collect the errors telemetry.

Name	Type	Description
Error type	String	The error's name, if one exists. For more information, see Error.prototype.name. Some browsers might not support error types.
Error message	String	The error's message. For more information, see Error.prototype.message. If the error field does not exist, this is the message of the error event. For more information, see ErrorEvent. Error messages might not be consistent across different browsers.
Stack trace	String	The error's stack trace, if one exists, truncated to 150 characters. For more information, see Error.prototype.stack. Some browsers might not support stack traces.

DOM event schema

Document object model (DOM) events generated by the agent contain the following properties. These events are not collected by default. They are collected only if you activate the interactions telemetry. For more information, see the CloudWatch RUM web client documentation.

Name	Type	Description
Event	String	The type of DOM event, such as click, scroll, or hover. For more information, see Event reference.
Element	String	The DOM element type

Name	Type	Description
Element ID	String	If the element that generated the event has an ID, this property stores that ID. For more information, see Element.id.
CSSLocator	String	The CSS locator used to identify the DOM element.
InteractionId	String	A unique ID for the interaction between the user and the UI.

Navigation event schema

Navigation events are collected only if the app monitor has performance telemetry activated.

Navigation events use Navigation timing Level 1 and Navigation timing Level 2 APIs. Level 2 APIs are not supported on all browsers, so these newer fields are optional.

> **Note**
>
> Timestamp metrics are based on DOMHighResTimestamp. With Level 2 APIs, all timings are by default relative to the `startTime`. But for Level 1, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Navigation events contain the following properties.

Name	Type	Description	Notes
initiatorType	String	Represents the type of resource that initiated the performance event.	**Value:** "navigation" **Level 1:** "navigation"

Name	Type	Description	Notes
			Level 2: entryData .initiatorType

Name	Type	Description	Notes
navigationType	String	Represents the type of navigation. This attribute is not required.	**Value:** The value must be one of the following: • `navigate` is a navigation started by choosing a link, entering a URL in a browser's address bar, form submission, or initializing through a script operation other than `reload` or `back_forward` . • `reload` is a navigation through the browser's reload operation or

Name	Type	Description	Notes
			`location.reload()`. • `back_forward` is a navigation through the browser's history traversal operation. • `prerender` is a navigation initiated by a prerender hint. For more information, see Prerender.

Name	Type	Description	Notes
startTime	Number	Indicates when the event is triggered.	**Value:** 0 **Level 1:** entryData.navigationStart - entryData.navigationStart **Level 2:** entryData.startTime

Name	Type	Description	Notes
unloadEventStart	Number	Indicates the time when the previous document in the window began to unload after the `unload` event was thrown.	**Value:** If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. **Level 1:** `entryData.unloadEventStart > 0 ? entryData.unloadEventStart - entryData.navigationStart : 0` **Level 2:** entryData.unloadEventStart

Name	Type	Description	Notes
promptForUnload	Number	The time taken to unload the document. In other words, the time between `unloadEventStart` and `unloadEventEnd`. `UnloadEventEnd` represents the moment in milliseconds when the unload event handler finishes.	**Value:** If there is no previous document or if the previous document or one of the needed redirects is not of the same origin, the value returned is 0. **Level 1:** entryData.unloadEventEnd - entryData.unloadEventStart **Level 2:** entryData.unloadEventEnd - entryData.unloadEventStart

Name	Type	Description	Notes
redirectCount	Number	A number representing the number of redirects since the last non-redirect navigation under the current browsing context. This attribute is not required.	**Value:** If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0. **Level 1:** Not available **Level 2:** entryData.redirectCount

Name	Type	Description	Notes
redirectStart	Number	The time when the first HTTP redirect starts.	**Value:** If there is no redirect or if there is any redirect that is not of the same origin as the destination document, the value returned is 0. **Level 1:** `entryData.redirectStart > 0 ? entryData.redirectStart - entryData.navigationStart : 0` **Level 2:** entryData.redirectStart

Name	Type	Description	Notes
redirectTime	Number	The time taken for the HTTP redirect. This is the difference between `redirectStart` and `redirectEnd` .	**Level 1:** : entryData .redirectEnd - entryData .redirectStart **Level 2:** : entryData .redirectEnd - entryData .redirectStart

Name	Type	Description	Notes
workerStart	Number	This is a property of the `PerformanceResourceTiming` interface. It marks the beginning of worker thread operation. This attribute is not required.	**Value:** If a Service Worker thread is already running, or immediately before starting the Service Worker thread, this property returns the time immediately before dispatching `FetchEvent`. It returns 0 if the resource is not intercepted by a Service Worker. **Level 1:** Not available **Level 2:** entryData.workerStart

Name	Type	Description	Notes
workerTime	Number	If the resource is intercepted by a Service Worker, this returns the time required for worker thread operation. This attribute is not required.	**Level 1:** Not available **Level 2:** `entryData.workerStart > 0 ? entryData.fetchStart - entryData.workerStart : 0`
fetchStart	Number	The time when the browser is ready to fetch the document using an HTTP request. This is before checking any application cache.	**Level 1:** `: entryData.fetchStart > 0 ? entryData.fetchStart - entryData.navigationStart : 0` **Level 2:** entryData.fetchStart

Name	Type	Description	Notes
domainLookupStart	Number	The time when the domain lookup starts.	**Value:** If a persistent connection is used or if the information is stored in a cache or local resource, the value will be the same as `fetchStart`. **Level 1:** `entryData.domainLookupStart > 0 ? entryData.domainLookupStart - entryData.navigationStart : 0` **Level 2:** entryData.domainLookupStart

Name	Type	Description	Notes
dns	Number	The time required for domain lookup.	**Value:** If the resources and DNS records are cached, the expected value is 0. **Level 1:** entryData.domainLookupEnd - entryData.domainLookupStart **Level 2:** entryData.domainLookupEnd - entryData.domainLookupStart
nextHopProtocol	String	A string representing the network protocol used to fetch the resource. This attribute is not required.	**Level 1:** Not available **Level 2:** entryData.nextHopProtocol

Name	Type	Description	Notes
connectStart	Number	The time immediately before the user agent starts establishing the connection to the server to retrieve the document.	**Value:** If an RFC2616 persistent connection is used, or if the current document is retrieved from relevant application caches or local resources, this attribute returns the value of `domainLookupEnd`. **Level 1:** `entryData.connectStart > 0 ? entryData.connectStart - entryData.navigationStart : 0` **Level 2:** entryData.connectStart

Name	Type	Description	Notes
connect	Number	Measures the time required to establish the transport connections or to perform SSL authentication. It also includes the blocked time that is taken when there are too many concurrent requests issued by the browser.	**Level 1:** entryData .connectEnd - entryData .connectStart **Level 2:** entryData .connectEnd - entryData .connectStart
secureConnectionStart	Number	If the URL scheme of the current page is "https", this attribute returns the time immediately before the user agent starts the handshake process to secure the current connection. It returns 0 if HTTPS is not used. For more information about URL schemes, see URL representation.	**Formula:** entryData .secureConnectionStart

Name	Type	Description	Notes
tlsTime	Number	The time taken to complete an SSL handshake.	**Level 1:** `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0` **Level 2:** `entryData.secureConnectionStart > 0 ? entryData.connectEnd - entryData.secureConnectionStart : 0`

Name	Type	Description	Notes
requestStart	Number	The time immediately before the user agent starts requesting the resource from the server, or from relevant application caches, or from local resources.	**Level 1:** `: entryData.requestStart > 0 ? entryData.requestStart - entryData.navigationStart : 0` **Level 2:** entryData.requestStart
timeToFirstByte	Number	The time taken to receive the first byte of information after a request is made. This time is relative to the `startTime` .	**Level 1:** entryData.responseStart - entryData.requestStart **Level 2:** entryData.responseStart - entryData.requestStart

Name	Type	Description	Notes
responseStart	Number	The time immediately after the user agent's HTTP parser receives the first byte of the response from the relevant application caches, or from local resources, or from the server.	**Level 1:** `entryData.responseStart > 0 ? entryData.responseStart - entryData.navigationStart : 0` **Level 2:** entryData.responseStart

Name	Type	Description	Notes
responseTime	String	The time taken to receive a complete response in the form of bytes from the relevant application caches, or from local resources, or from the server.	**Level 1:** `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0` **Level 2:** `entryData.responseStart > 0 ? entryData.responseEnd - entryData.responseStart : 0`

Name	Type	Description	Notes
domInteractive	Number	The time when the parser finished its work on the main document, and the HTML DOM is constructed. At this time, its `Document.readyState` changes to "interactive" and the corresponding `readystatechange` event is thrown.	**Level 1:** `entryData.domInteractive > 0 ? entryData.domInteractive - entryData.navigationStart : 0` **Level 2:** entryData.domInteractive

Name	Type	Description	Notes
domContentLoadedEventStart	Number	Represents the time value equal to the time immediately before the user agent fires the DOMContentLoaded event at the current document. TheDOMContentLoaded event fires when the initial HTML document has been completely loaded and parsed. At this time, the main HTML document has finished parsing, the browser begins constructing the render tree, and subresources still have to be loaded. This does not wait for style sheets, images, and subframes to finish loading.	**Level 1:** `entryData.domContentLoadedEventStart > 0 ? entryData.domContentLoadedEventStart - entryData.navigationStart : 0` **Level 2:** entryData.domContentLoadedEventStart

Name	Type	Description	Notes
domContentLoaded	Number	This start and end time of render tree construction is marked by the `domContentLoadedEventStart` and `domContentLoadedEventEnd`. It lets CloudWatch RUM track execution. This property is the difference between `domContentLoadedStart` and `domContentLoadedEnd`. During this time, DOM and CSSOM are ready. This property waits on script execution, except for asynchronous and dynamically created scripts. If the scripts depend on style sheets, `domContentLoaded` waits on the style sheets, too. It does not wait on images. **Note** The actual values of `domContentLoadedStart` and `domContentLoadedEnd` approximate to `domContentLoaded` in Google Chrome's Network panel. It indicates HTML DOM + CSSOM render tree construction time from the beginning of the page loading process. In the case of navigation metrics, the `domContentLoaded` value represents the difference between start and end values, which is the time required for downloading subresources and render-tree construction only.	**Level 2:** entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart **Level 2:** entryData.domContentLoadedEventEnd - entryData.domContentLoadedEventStart

Name	Type	Description	Notes
domComplete	Number	The time immediately before the browser sets the current document readiness of the current document to complete. At this point, the loading of subresources, such as images, is complete. This includes the time taken for downloading blocking content such as CSS and synchronous JavaScript. This approximates to `loadTime` in Google Chrome's Network panel.	**Level 1:** `entryData.domComplete > 0 ? entryData.domComplete - entryData.navigationStart : 0` **Level 2:** entryData.domComplete
domProcessingTime	Number	The total time between the response and the load event start.	**Level 1:** entryData.loadEventStart - entryData.responseEnd **Level 2:** entryData.loadEventStart - entryData.responseEnd

Name	Type	Description	Notes
loadEvent Start	Number	The time immediately before the `load` event of the current document is fired.	**Level 1:** `entryData.loadEventStart > 0 ? entryData.loadEventStart - entryData.navigationStart : 0` **Level 2:** entryData.loadEventStart
loadEvent Time	Number	The difference between `loadEventStart` and `loadEventEnd` . Additional functions or logic waiting for this load event will be fired during this time.	**Level 1:** entryData.loadEventEnd - entryData.loadEventStart **Level 2:** entryData.loadEventEnd - entryData.loadEventStart

Name	Type	Description	Notes
duration	String	Duration is the total page load time. It records the timing for downloading the main page and all of its synchronous subresources, and also for rendering the page. Asynchronous resources such as scripts continue to download later. This is the difference between the `loadEventEnd` and `startTime` properties.	**Level 1:** entryData.loadEventEnd - entryData.navigationStart **Level 2:** entryData.duration
headerSize	Number	Returns the difference between `transferSize` and `encodedBodySize`. This attribute is not required.	**Level 1:** Not available **Level 2:** entryData.transferSize - entryData.encodedBodySize **Level 2:** entryData.transferSize - entryData.encodedBodySize

Name	Type	Description	Notes
compressionRatio	Number	The ratio of `encodedBodySize` and `decodedBodySize`. The value of `encodedBodySize` is the compressed size of the resource excluding the HTTP headers. The value of `decodedBodySize` is the decompressed size of the resource excluding the HTTP headers. This attribute is not required.	**Level 1:** Not available. **Level 2:** `entryData.encodedBodySize > 0 ? entryData.decodedBodySize / entryData.encodedBodySize : 0`
navigationTimingLevel	Number	The navigation timing API version.	**Value:** 1 or 2

Resource event schema

Resource events are collected only if the app monitor has performance telemetry activated.

Timestamp metrics are based on The DOMHighResTimeStamp typedef. With Level 2 APIs, by default all timings are relative to the `startTime`. But for Level 1 APIs, the `navigationStart` metric is subtracted from timestamp metrics to obtain relative values. All timestamp values are in milliseconds.

Resource events generated by the agent contain the following properties.

Name	Type	Description	Notes
targetUrl	String	Returns the resource's URL.	**Formula:** entryData .name
initiatorType	String	Represents the type of resource that initiated the performance resource event.	**Value:** "resource" **Formula:** entryData .initiatorType
duration	String	Returns the difference between the `responseEnd` and `startTime` properties. This attribute is not required.	**Formula:** entryData .duration
transferSize	Number	Returns the size (in octets) of the fetched resource, including the response header fields and the response payload body. This attribute is not required.	**Formula:** entryData .transferSize
fileType	String	Extensions derived from the target URL pattern.	

Largest contentful paint event schema

Largest contentful paint events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

First input delay event

First input delay events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

Cumulative layout shift event

Cumulative layout shift events contain the following properties.

These events are collected only if the app monitor has performance telemetry activated.

Name	Description		
Value	For more information, see Web Vitals.		

HTTP event

HTTP events can contain the following properties. It will contain either a `Response` field or an `Error` field, but not both.

These events are collected only if the app monitor has HTTP telemetry activated.

Name	Description
Request	The request field includes the following: • The `Method` field, which can have values such as GET, POST, and so on. • The URL

Name	Description
Response	The response field includes the following: • Status, such as 2xx, 4xx, or 5xx • Status text
Error	The error field includes the following: • Type • Message • File name • Line number • Column number • Stack trace

X-Ray trace event schema

These events are collected only if the app monitor has X-Ray tracing activated.

For information about X-Ray trace event schemas, see AWS X-Ray segment documents.

Route change timing for single-page applications

In a traditional multi-page application, when a user requests for new content to be loaded, the user is actually requesting a new HTML page from the server. As a result, the CloudWatch RUM web client captures the load times using the regular performance API metrics.

However, single-page web applications use JavaScript and Ajax to update the interface without loading a new page from the server. Single-page updates are not recorded by the browser timing API, but instead use route change timing.

CloudWatch RUM supports the monitoring of both full page loads from the server and single-page updates, with the following differences:

- For route change timing, there are no browser-provided metrics such as `tlsTime`, `timeToFirstByte`, and so on.
- For route change timing, the `initiatorType` field will be `route_change`.

The CloudWatch RUM web client listens to user interactions that may lead to a route change, and when such a user interaction is recorded, the web client records a timestamp. Then route change timing will begin if both of the following are true:

- A browser history API (except browser forward and back buttons) was used to perform the route change.
- The difference between the time of route change detection and latest user interaction timestamp is less than 1000 ms. This avoids data skew.

Then, once route change timing begins, that timing completes if there are no ongoing AJAX requests and DOM mutations. Then the timestamp of the latest completed activity will be used as the completion timestamp.

Route change timing will time out if there are ongoing AJAX requests or DOM mutations for more than 10 seconds (by default). In this case, the CloudWatch RUM web client will no longer record timing for this route change.

As a result, the duration of a route change event is calculated as the following:

```
(time of latest completed activity) - (latest user interaction timestamp)
```

Manage your applications that use CloudWatch RUM

Use the steps in these sections to manage your applications' use of CloudWatch RUM.

Topics

- How do I find a code snippet that I've already generated?
- Editing your CloudWatch RUM app monitor settings
- Stopping using CloudWatch RUM or deleting an app monitor

How do I find a code snippet that I've already generated?

To find a CloudWatch RUM code snippet that you've already generated for an application, follow these steps.

To find a code snippet that you've already generated

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Navigate to the **Configuration** tab, **Code snippets** section.
4. Choose **Copy/Download** for your respective instrumentation.

Editing your CloudWatch RUM app monitor settings

To change an app monitor's settings, follow these steps. You can change any settings except the app monitor name.

To edit how your application uses CloudWatch RUM

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose the button next to the name of the application, and then choose **Actions**, **Edit**.
4. Change any settings except the application name. For more information about the settings, see Creating a CloudWatch RUM app monitor for a web application.
5. When finished, choose **Save**.

   Changing the settings changes the code snippet. You must now paste the updated code snippet into your application.
6. After the code snippet is created, choose **Copy to clipboard** or **Download**, and then choose **Done**.

   To start monitoring with the new settings, you insert the code snippet into your application.

Stopping using CloudWatch RUM or deleting an app monitor

To stop using CloudWatch RUM with an application, remove the code snippet that RUM generated from your application's code.

To delete a RUM app monitor, follow these steps.

To delete an app monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Application Signals**, **RUM**.
3. Choose the button next to the name of the application, and then choose **Actions**, **Delete**.

4. In the confirmation box, enter **`Delete`** and then choose **Delete**.
5. If you haven't done so already, delete the CloudWatch RUM code snippet from your application's code.

Troubleshooting CloudWatch RUM

This section contains tips to help you troubleshoot CloudWatch RUM.

There is no data for my application

First, make sure that the code snippet has been correctly inserted into your application. For more information, see Inserting the CloudWatch app monitor code snippet into your application.

If that is not the issue, then maybe there has been no traffic to your application yet. Generate some traffic by accessing your application the same way that a user would.

Data has stopped being recorded for my application

Your application might have been updated and now no longer contains a CloudWatch RUM code snippet. Check your application code.

Another possibility is that someone may have updated the code snippet but then didn't insert the updated snippet into the application. Find the current correct code snippet by following the directions in How do I find a code snippet that I've already generated? and compare it to the code snippet that is pasted into your application.

Generative AI observability

With Amazon CloudWatch, you can observe generative AI workloads, including Amazon Bedrock AgentCore agents, and gain insights into AI performance, health, and accuracy. CloudWatch provides pre-configured views into latency, usage, and errors of your AI workloads, allowing you to detect issues faster in components like models and agents. End-to-end prompt tracing helps you quickly identify issues in components such as knowledge bases, tools, and models. CloudWatch's AI monitoring capabilities are compatible with popular generative AI orchestration frameworks such as AWS Strands, LangChain, and LangGraph, offering flexibility with your choice of framework.

CloudWatch generative AI observability enables you to:

- Assess AI application quality and accuracy at scale through automated monitoring, reducing manual review requirements by capturing model outputs, response quality metrics, and end-user interactions
- Monitor model invocations, Agents (managed, self-hosted, and third-party), knowledge bases, guardrails, and tools
- Progress from agent experimentation to production of innovative generative AI applications while ensuring superior quality, performance, and reliability. For more information, see What is Amazon Bedrock AgentCore?
- Identify source of errors quickly using end-to-end prompt tracing, curated metrics, and logs
- Troubleshoot issues across your entire generative AI application and underlying infrastructure, leveraging existing CloudWatch observability tools such as Application Signals, Alarms, Dashboards, Sensitive data protection, and Logs Insights
- Access prompt traces while using Amazon Bedrock, and send structured traces of third-party models to CloudWatch using ADOT SDK. For information about adding observability to your Amazon Bedrock AgentCore agent or tool, see Amazon Bedrock AgentCore

CloudWatch generative AI observability provides two pre-built capabilities:

> **Note**
>
> You can use the **Model Invocation** dashboard by using any models for inference in Amazon Bedrock.

- **Model Invocations** – Detailed metrics dashboard on model usage, token consumption, and a curated invocation logs table to view detailed input and output content of model inferences
- **Amazon Bedrock AgentCore agents** – Performance and decision metrics for primitives of Amazon Bedrock AgentCore such as Agents, Memory, Built-in Tools, Gateways, and Identity

Key metrics available in these dashboards include:

- Total and average invocations
- Token usage (total, average per query, input, output)
- Latency (average, P90, P99)
- Error rates and throttling events
- Cost attribution by application, user role, or specific user

Topics

- Model Invocations
- Amazon Bedrock AgentCore
- Monitor AI coding agents with Amazon CloudWatch Coding Agent Insights

Model Invocations

CloudWatch generative AI observability allows you to monitor Model Invocations performance. You can track metrics such as invocation count, token usage, and errors using out-of-box views. For detailed visibility into invocation content, such as inputs and outputs, enable **Bedrock Invocation logging** and send the logs to CloudWatch. For more information, see Set up a CloudWatch Logs destination and Help protect sensitive log data with masking .

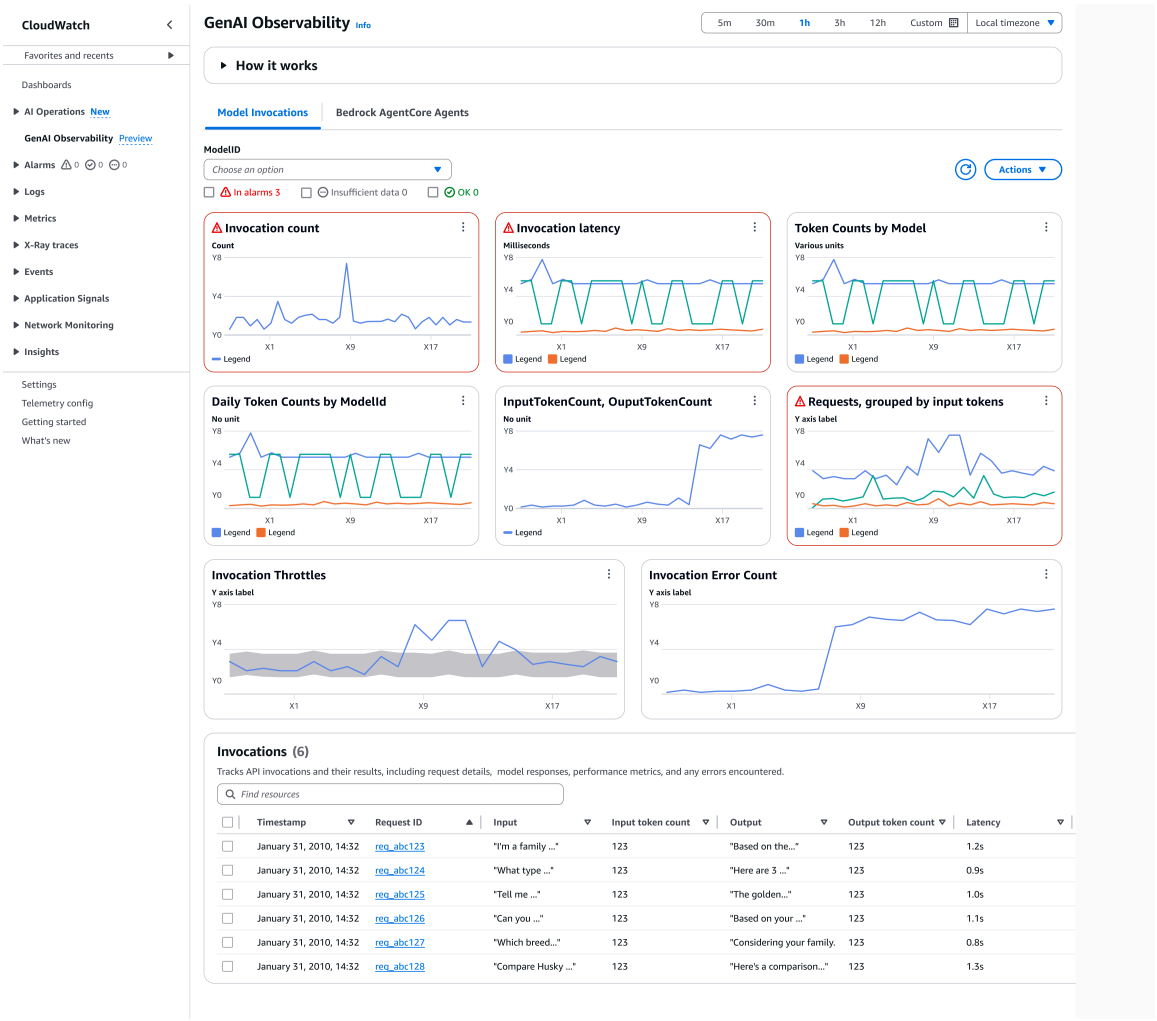
Enabling model invocation in Amazon Bedrock

> **Note**
>
> You must enable Model invocation logging in Amazon Bedrock to view the invocations.

To enable model invocation logging in Amazon Bedrock, follow these steps.

1. Open the Amazon Bedrock console at https://console.aws.amazon.com/bedrock/.
2. Choose **Settings**.
3. Under **Model invocation logging**, select **Model invocation logging**.
4. Choose the required data types to include in the logs. Choose to send the logs to CloudWatch Logs only or both Amazon S3 and CloudWatch Logs if you are already publishing to Amazon S3.
5. Under the CloudWatch Logs configurations, create log group name and select the appropriate service roles.
6. Choose the required data types to include in the logs.
7. Choose **Save settings**

   You can view the pre-configured dashboards automatically when you start using Amazon Bedrock invocations. After enabling `Model Invocation logging`, you can view the default dashboards and access the invocation table below them.

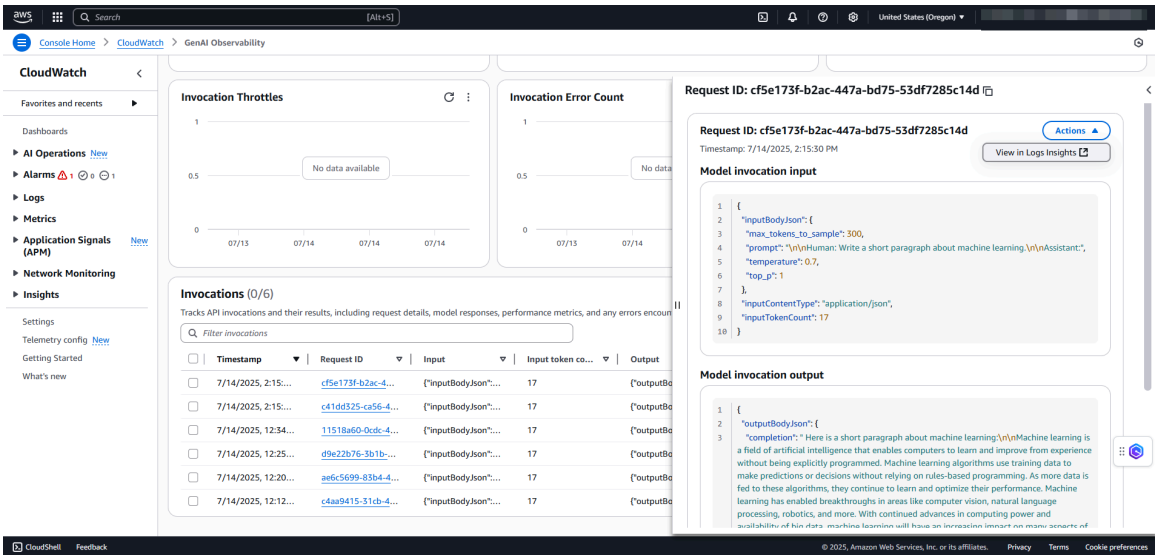


- **Invocation count** – Number of successful requests to the Converse, ConverseStream, InvokeModel, and InvokeModelWithResponseStream API operations
- **Invocation latency** – Latency of the invocations
- **Token Counts by Model** – Token counts by model delineated by input token counts and output token counts
- **Daily Token Counts by ModelID** – Daily total token counts by model ID
- **InputTokenCount, OutputTokenCount** – Total number of tokens in the input and output in this account across selected models
- **Requests, grouped by input tokens** – Number of requests grouped by input tokens into 6 ranges. Each line represent the number of requests that fall into a particular range
- **Invocation Throttles** – Number of invocations the system throttled. The number of throttles you see will depend on your retry settings in the SDK. For more information, see Retry behavior in the AWS SDKs and Tools Reference Guide

- **Invocation Error Count** – Count of invocations that result in server-side and client-side errors

To use the model invocation dashboard, follow these steps.

1. Hover over any metric graph to view invocation details. You can choose the **Alarm** icon to setup `Alarms` to monitor application quality and performance.
2. Under the **ModelID** drop-down, you can select a model ID to view the corresponding metrics.
3. Select **View in CloudWatch metrics** to view the dashboard metrics under CloudWatch.
4. Select **Period override** to adjust the metrics timeframe (for example, 1 minute, 1 hour, or 6 hours).
5. Under **Invocations**, choose **Request ID** to view the details of the request. You can view the model invocation input and output details on the right-pane.



On the **Request ID** page, under **Actions** drop-down, choose **View in Logs Insights** to view the logs in CloudWatch. For more information, see Analyzing log data with CloudWatch Logs Insights.

Amazon Bedrock AgentCore

CloudWatch provides curated observability views for Amazon Bedrock AgentCore. These views help you monitor the operational health of your managed agents. You can use observability views and prompt tracing to debug and monitor AI agent performance. To monitor your AI agents, enable **Observability** in Amazon Bedrock AgentCore. Amazon Bedrock AgentCore includes a collection of

modular services such as agents, memory, tools, and gateway. You can use these services together or independently to build your AI applications.

For more information on the modular services, see Amazon Bedrock AgentCore.

Topics

- Getting started
- Agents
- Memory
- Built-in tools
- Gateways
- Identity

Getting started

Amazon Bedrock AgentCore provides built-in metrics, logs, and traces to monitor the performance of your AgentCore modular services. You can view this data in Amazon CloudWatch. To access the full range of observability data from all AgentCore module services, instrument your code using the AWS Distro for OpenTelemetry (ADOT) SDK.

Add observability to your agentic resources

Before you begin, enable CloudWatch Transaction Search. For more information, see Enable Transaction Search.

Enable observability for AgentCore Runtime hosted agents

You can host your agents on AgentCore Runtime, a secure, serverless runtime purpose-built for deploying and scaling dynamic AI agents and tools. AgentCore Runtime supports any open-source framework including LangGraph, CrewAI, Strands Agents, any protocol, and any model.

To enable observability for AgentCore Runtime hosted agents, see Configure custom observability.

For a step-by-step tutorial, see Enabling observability for AgentCore Runtime hosted agents.

Enable observability for non-AgentCore hosted agents

You can host your agents outside of AgentCore and bring your observability data into CloudWatch for end-to-end monitoring in one location.

To enable observability for non-AgentCore Runtime hosted agents, see Configure third-party observability.

For a step-by-step tutorial, see Enabling observability for non-AgentCore hosted agents.

Enable observability for AgentCore memory, gateway, and built-in tool resources

You can gain visibility into the metrics and traces of AgentCore modular services. For more information, see Configure CloudWatch observability.

Enable AgentCore Evaluations

You can gain visibility into AgentCore Evaluations. AgentCore Evaluations provide capabilities to monitor and assess the performance, quality, and reliability of your AI agents. To enable observability for AgentCore Evaluations, see AgentCore evaluations.

View observability data in CloudWatch

After you enable observability for your agentic resources, you can view the collected data in CloudWatch.

View the GenAI Observability dashboard

1. Open the CloudWatch console.
2. Under the GenAI Observability dashboard, view data related to model invocations and agents on Amazon Bedrock AgentCore.
3. In the Amazon Bedrock AgentCore sub-menu, you can choose the following views:

   - **Agents View** – Lists all your agents, both on and off runtime. Choose an agent to view runtime metrics, sessions, traces, and evaluations specific to that agent
   - **Sessions View** – Navigate across all sessions associated with agents
   - **Traces View** – View traces and span information for agents. Choose a trace to explore the trace trajectory and timeline

View logs

1. Open the CloudWatch console.
2. In the navigation pane, expand **Logs** and choose **Log groups**.
3. Search for your agent's log group:

- Standard logs (stdout/stderr) – `/aws/bedrock-agentcore/runtimes/<agent_id>-<endpoint_name>/[runtime-logs] <UUID>`
- OTEL structured logs – `/aws/bedrock-agentcore/runtimes/<agent_id>-<endpoint_name>/runtime-logs`

View traces and spans

1. Open the CloudWatch console.
2. In the navigation pane, choose **Transaction Search**.
3. Navigate to `/aws/spans/default`.
4. Filter by service name or other criteria.
5. Choose a trace to view the detailed execution graph.

View metrics

1. Open the CloudWatch console.
2. In the navigation pane, choose **Metrics**.
3. Navigate to the **bedrock-agentcore** namespace.
4. Explore the available metrics.

Protect sensitive data

Amazon CloudWatch Logs uses data protection policies to identify sensitive data and define actions to protect that data. You use data identifiers to select the sensitive data of interest. Amazon CloudWatch Logs then detects the sensitive data using machine learning and pattern matching. You can define audit and masking operations to log sensitive data findings and mask sensitive data when viewing log events.

For more information, see Protecting sensitive log data with masking.

You can configure data protection for Amazon Bedrock AgentCore at the **account level** or at the **log group level**. With account level data protection, data protection rules are applied to all logs in your account. With log level data protection, data protection rules are can be applied to specific log groups in your account. This gives you granular control over how PII data is masked in your in your account.

To configure data protection at the account level

1. Open the Amazon CloudWatch console.
2. In the navigation pane, choose **Settings**.
3. Choose the **Logs** tab.
4. Choose **Configure the Data protection account policy**.
5. Specify the data identifiers that are relevant to your data.
   - To use a a predefined data identifier, in the **Managed data identifiers** drop-down, select the data identifiers that are relevant to your data.
   - To use a custom data identifier, choose **Add custom data identifier**, and then specify a name for the identifier and a Regex pattern for the data to protect.
6. (*Optional*) Choose a destination for the audit findings.
   - To send audit findings to a CloudWatch log, choose **Amazon CloudWatch Logs** and then select the destination log group.
   - To send audit findings to a Firehose stream, choose **Amazon Data Firehose** and then select the destination firehose stream.
   - To send audit findings to an Amazon S3 bucket, choose **Amazon S3** and then select the destination Amazon S3 bucket.
7. Choose **Activate data protection**.

To configure data protection at the log group level

1. Open the Amazon CloudWatch console.
2. In the navigation panel, choose **Logs**, **Log Management**.
3. Choose the **Log groups** tab, select the log group you want to enable data protection on, and then choose **Create data protection policy**.
4. Specify the data identifiers that are relevant to your data.
   - To use a a predefined data identifier, in the **Managed data identifiers** drop-down, select the data identifiers that are relevant to your data.
   - To use a custom data identifier, choose **Add custom data identifier**, and then specify a name for the identifier and a Regex pattern for the data to protect.
5. (*Optional*) Choose a destination for the audit findings.

- To send audit findings to a CloudWatch log, choose **Amazon CloudWatch Logs** and then select the destination log group.
- To send audit findings to a Firehose stream, choose **Amazon Data Firehose** and then select the destination firehose stream.
- To send audit findings to an Amazon S3 bucket, choose **Amazon S3** and then select the destination Amazon S3 bucket.

6. Choose **Activate data protection**.

Agents

You can use *Agents* to monitor agent performance, track their decision-making processes, analyze conversation flows, and troubleshoot issues through comprehensive metrics and tracing. This includes monitoring both Amazon Bedrock AgentCore managed agents and self-hosted or third-party agents that emit telemetry data to CloudWatch.

You can analyze various Agents and their associated interactions under **Agent view**, **Sessions view**, and **Traces view**. For more information, see Understand observability for agentic resources in AgentCore .

Topics

- Agent view
- Session view
- Traces view

Agent view

The *Agent view* provides a curated dashboard for your account's agents. You can view data from agents hosted on AWS native services like AgentCore Runtime, Lambda, or Amazon EC2. The view also displays agents that emit telemetry to CloudWatch.

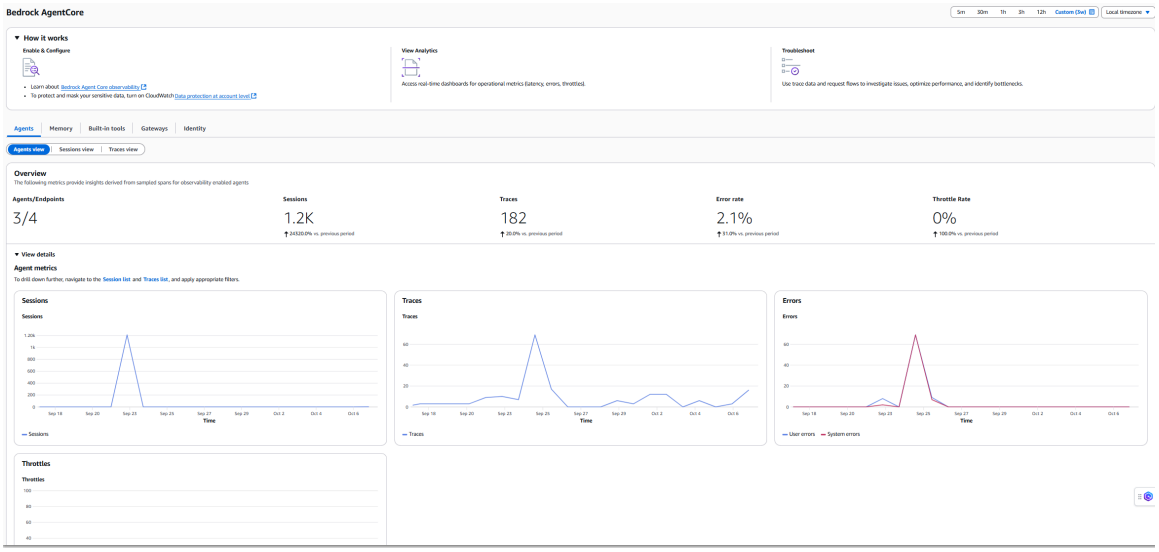
Overview

The metrics and dashboards show data from sampled agent spans. For information about agent spans, see Spans.

The following Agent metrics are supported:

- Agents/Endpoints – Number of agents and aliases instrumented and emitting spans
- Sessions – Number of sessions created by instrumented agents emitting spans. A session is similar to a conversation and contains the broad context
- Traces – Number of traces created by instrumented agents emitting spans. A trace is a individual request-response cycle within a session
- Error rate – Percentage of errors in agent interactions
- Throttle rate – Percentage of throttled agent interactions

Choose **View details** to see the Agent metrics in graphs.



Runtime metrics

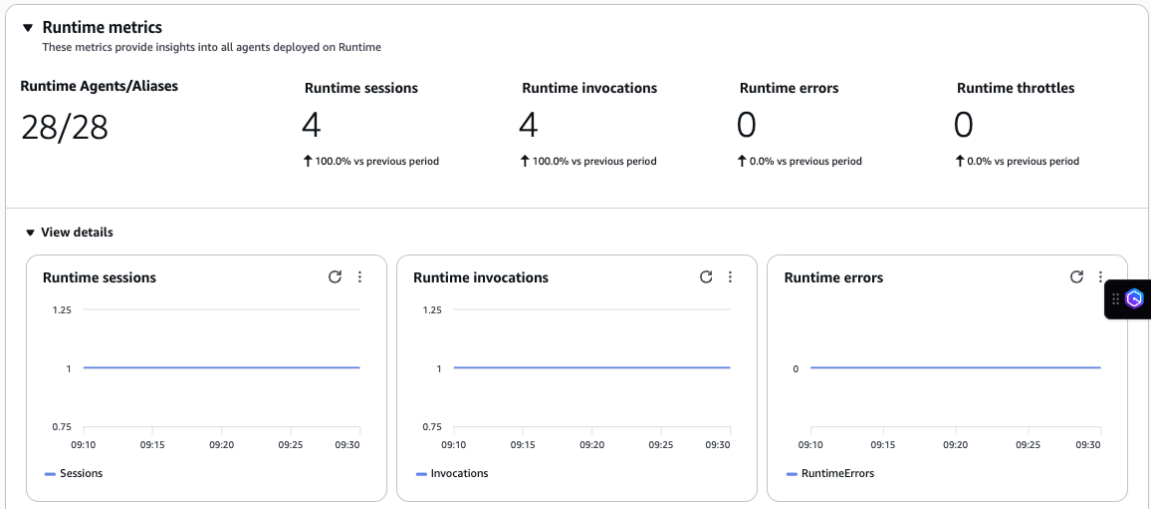
The Runtime metrics and dashboards display data from the Runtime primitive. Using this primitive, you can host your agents on the Amazon Bedrock AgentCore runtime. For more information, see Creating an AgentCore Runtime .

AgentCore Runtime supports these metrics

- Runtime Agents/Aliases – Tracks number of agents and aliases hosted on AgentCore Runtime
- Runtime sessions – Tracks number of sessions created by agents running in AgentCore Runtime. A session is similar to a conversation and contains the broad context of the entire interaction flow. Useful for monitoring overall platform usage, capacity planning, and understanding user engagement patterns
- Runtime invocations – Total number of requests made to the Data Plane API. Each API call counts as one invocation, regardless of the request payload size or response status

- Runtime errors – The number of system and user errors. For system and user error definitions, see AgentCore provided runtime metrics
- Runtime throttles – The number of requests throttled by the service due to exceeding allowed TPS (Transactions Per Second). These requests return ThrottlingException with HTTP status code 429. Monitor this metric to determine if you need to review your service quotas or optimize request patterns

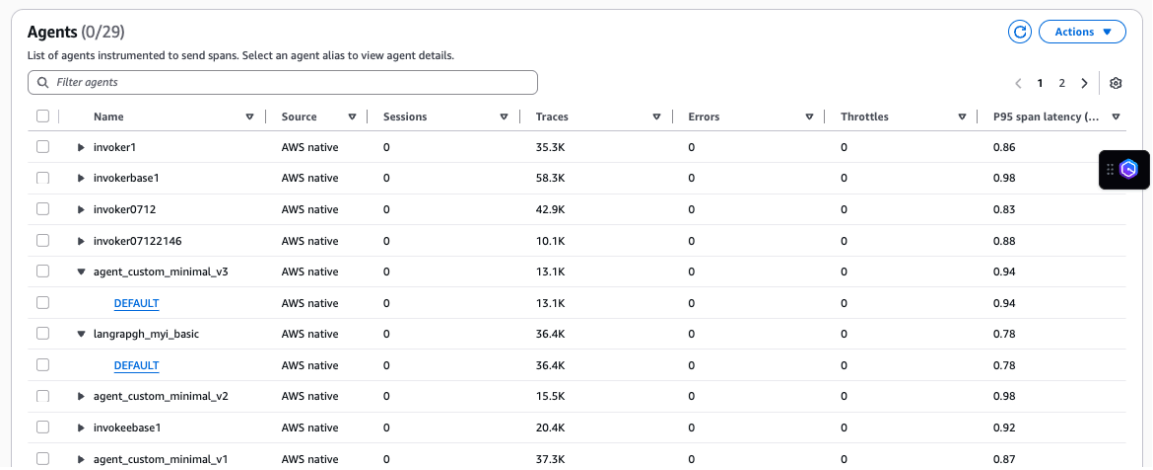
View metric changes over time in the default dashboard. Expand **View details** to display metric graphs.



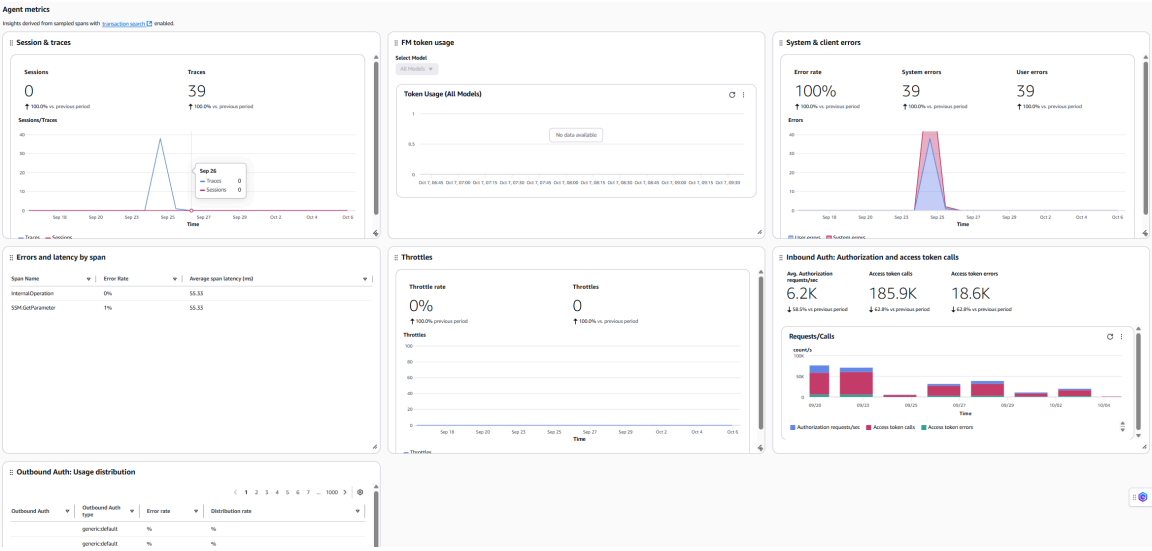
Agents

Agents are components that collect and send monitoring data from your applications. The Agents table displays all agents configured in your account. These agents can be hosted on AWS native services like AgentCore Runtime, Lambda, or Amazon EC2. The table also displays other agents that are instrumented to emit telemetry to CloudWatch.

You can use **Filter agents** to find a specific agent that you want to deep dive or you can also use the column names to sort the agents to find the required agent. Select the gear icon to show or hide additional columns.



You can view the details of the Agent by expanding the agent name.



Agent details- Overview

The Overview tab displays automatic dashboards for your agent metrics. These metrics come from sampled spans and Runtime metrics (when the agent uses AgentCore Runtime).

The **Evaluators** dashboard includes insights derived from spans with evaluations enabled.

- Top deltas in evaluator scores — Shows the agent evaluators that experienced the most change since the last period based on the time period you selected.
- Evaluation configuration metrics — Show the operational status metrics for the agent evaluators, including the number of times the evaluations were executed and the number of errors encountered.

To edit an evaluation configuration using the Amazon Bedrock AgentCore console, click the link in the **Evaluator** or **Evaluation configuration** column. To review the evaluator results, click a score in the **Avg. score** column. To view all evaluations for the agent, choose the **Evaluations** tab. For more information, see Agent details - Evaluations.

The **Agent metrics** dashboard include metrics which are derived from sampled spans:

- Sessions and Traces – Count of sessions and traces for this agent
- FM token usage – Total count of Foundational Model token consumption. You can filter the chart into a particular Foundational Model
- System and client errors – Count of system errors during request processing. High levels of server-side errors can indicate potential infrastructure or service issues that require investigation. Client errors are errors resulting from invalid requests. High levels of client-side errors can indicate issues with request formatting or permissions
- Errors and latency by span – The error rates and latency by a particular span. Note that a span can appear in many agents
- Throttles – Number of requests throttled by the service due to exceeding allowed TPS (Transactions Per Second)
- Inbound Auth:Authorization and access token calls – Number of incoming authentication requests processed by the agent, including authorization checks and access token validations from external clients or services
- Outbound Auth:Usage distribution – Distribution pattern of outbound authentication methods used by the agent, showing the frequency and types of authentication mechanisms employed when accessing external services

The **Runtime metrics** dashboard includes metrics that AgentCore Runtime automatically generates:

- Runtime sessions and invocations – Count of sessions and invocations that this particular agent has generated while being hosted on Runtime
- Runtime latency – Latency of requests by agents hosted on Runtime
- Runtime throttles – Number of requests throttle by the service due to exceeding allowed TPS (Transactions Per Second)

Agent details - Sessions

An Agent can have several Sessions. View session in the *Sessions* tab. Use the **Filter sessions** or sort the columns to find the required session.

Choose the **Session ID** to view the session summary metrics and the list of traces belonging to that session. Session metrics include:

- Traces – Number of traces belonging to the sessions
- Server errors – Count of system errors during request processing. High levels of server-side errors can indicate potential infrastructure or service issues that require investigation
- Client errors – Client errors are errors resulting from invalid requests. High levels of client-side errors can indicate issues with request formatting or permissions
- Throttles – Number of requests throttled relevant to this session due to exceeding allowed TPS (Transactions Per Second)
- Sessions details – Meta data about the session such as start time, end time, and session ID

To analyze a list of Traces in a session, choose **Filter traces** to narrow down or sort the table columns to bubble up the particular Trace you want to investigate.

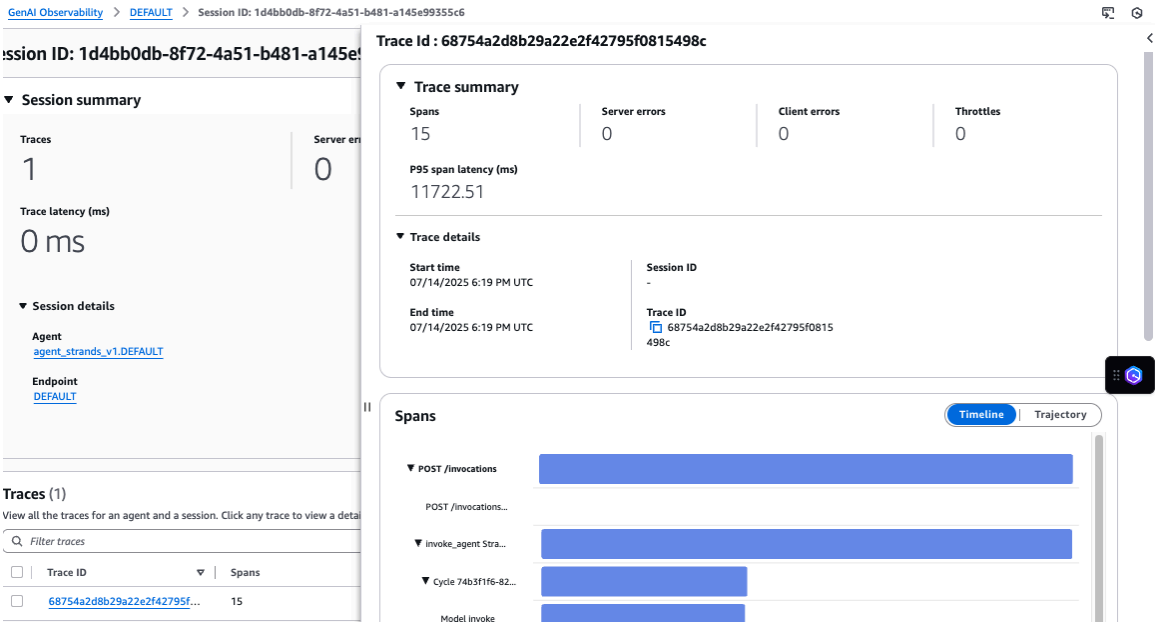
After you select a Trace, the right-pane displays the details of the Trace. For each Trace, you can see the Trace summary, Spans, and Trace content details.

Under **Trace summary**, you can view the following metrics:

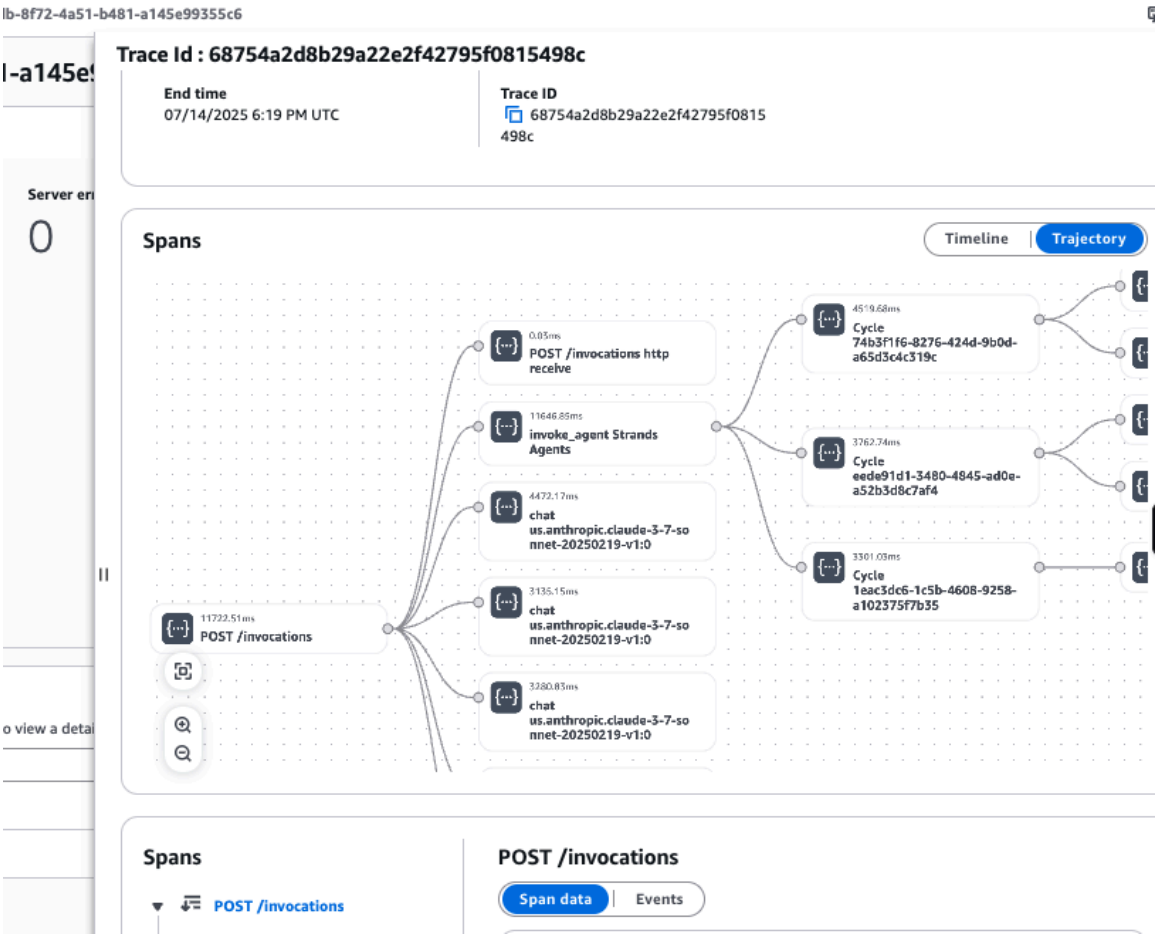
> **Note**
>
> Summary page fields are consistent across **Agent view**, **Sessions view**, and **Traces view**.

- Spans – Number of spans within a Trace
- Server errors – Count of system errors during request processing. High levels of server-side errors can indicate potential infrastructure or service issues that require investigation
- Client errors – Client errors are errors resulting from invalid requests. High levels of client-side errors can indicate issues with request formatting or permissions
- Throttles – Number of requests throttle relevant to this session due to exceeding allowed TPS (Transactions Per Second)

- P95 span latency – The 95-percentile latency of across all invocation of this particular span. Note that a span can be used across many agents
- Trace details – Meta data about the trace such as start time, end time, and Trace ID



Choose **Timeline** to view the duration of each span and to understand the span that took the longest and contributed to a slow response.

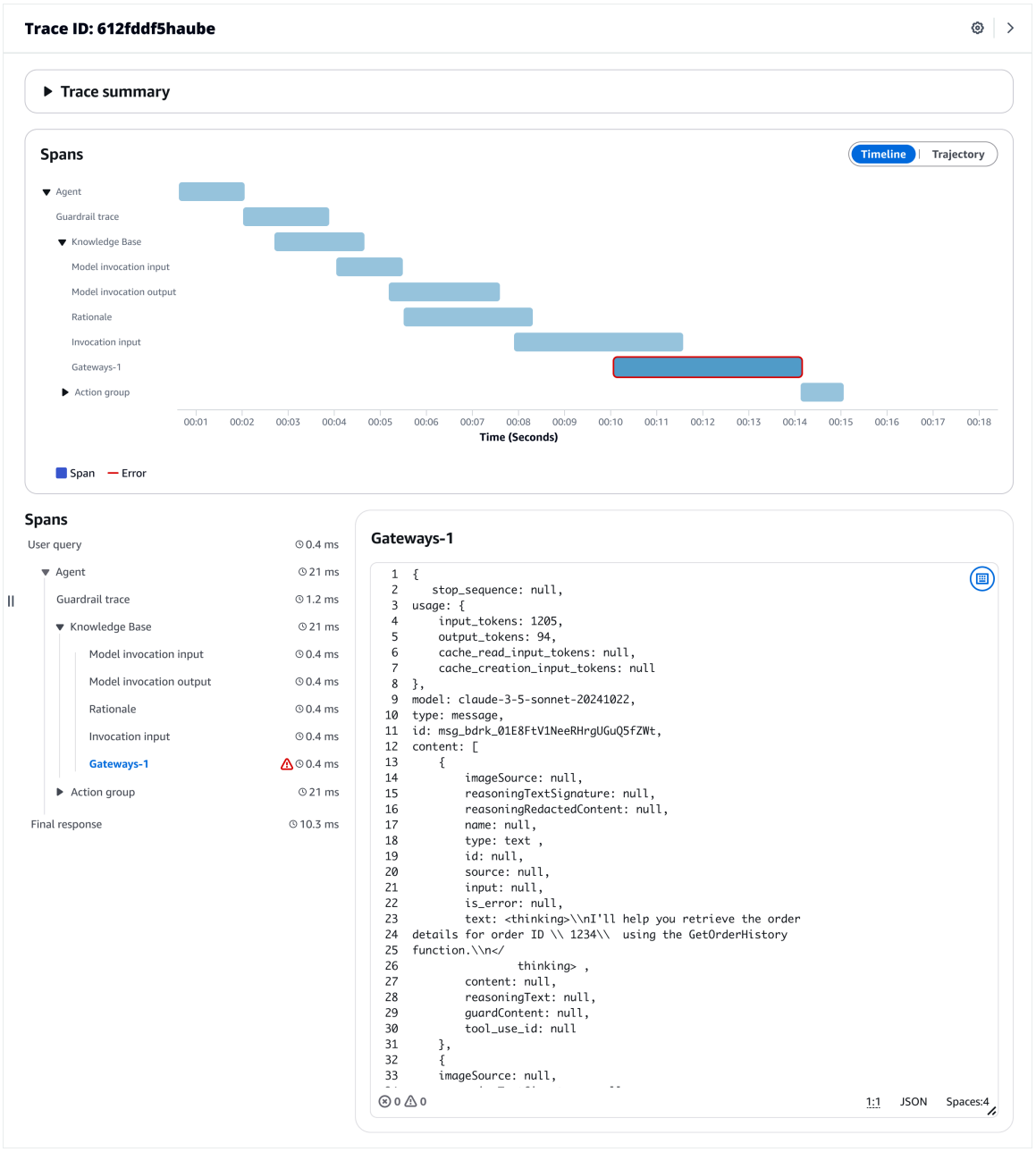


To analyze span relationships and subsequent calls choose **Trajectory** to understand the interconnected relationship of the spans and subsequent calls from these spans.

Under **Spans**, select an individual span event to review the span data in its original form. Review the span data in its original form. For granular troubleshooting, select the **Events** tab to examine model inputs and outputs.

Agent details - Traces

Each agent might have multiple traces. View trace details in the **Traces** tab. Choose **Filter traces** or sort the columns to find the required Trace.



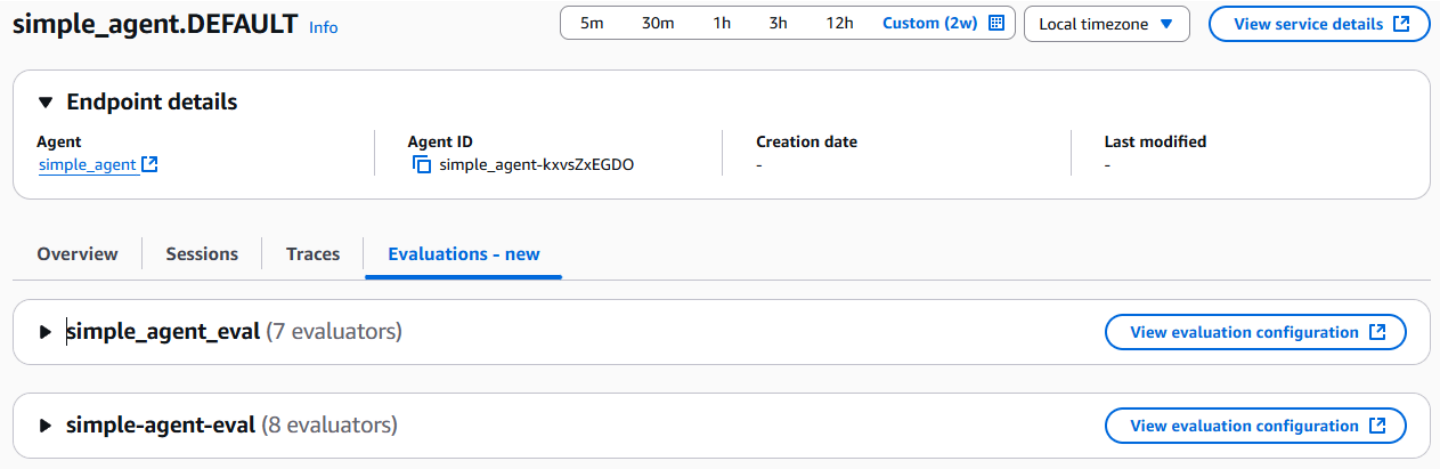
Agent details - Evaluations

Evaluations provides continuous quality monitoring metrics for your AI agents. You can use the information provided by the dashboard to assess the performance, quality, and reliability of your AI agents.

Instead of relying on simulated test cases, evaluations capture real user sessions and agent interactions, providing a comprehensive view of agent performance, from input to final output. With agent evaluations, you can define sampling rules to evaluate only a percentage of the sessions or traces, and then apply a variety of evaluators to asses and score an AI agent's

operational performance. The resulting assessments and scores are displayed in the Evaluations dashboard, allowing you to monitor trends, identify potential quality issues, set alarms, and investigate and diagnose potential issues.

The Evaluations dashboard lists all of the evaluations that have been enabled and configured for the selected agent. For more information about configuring evaluations for an agent, see AgentCore evaluations. You can expand each evaluation to view the sessions, traces, and spans that were evaluated.



Topics

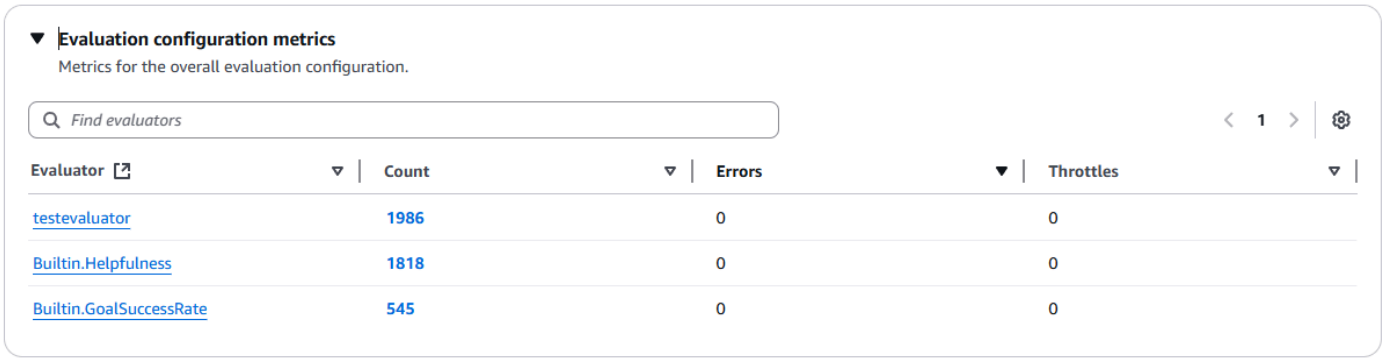
- Evaluations details
- Evaluations graphs
- Work with evaluation results

Evaluations details

For each evaluation, the dashboard includes the following sections:

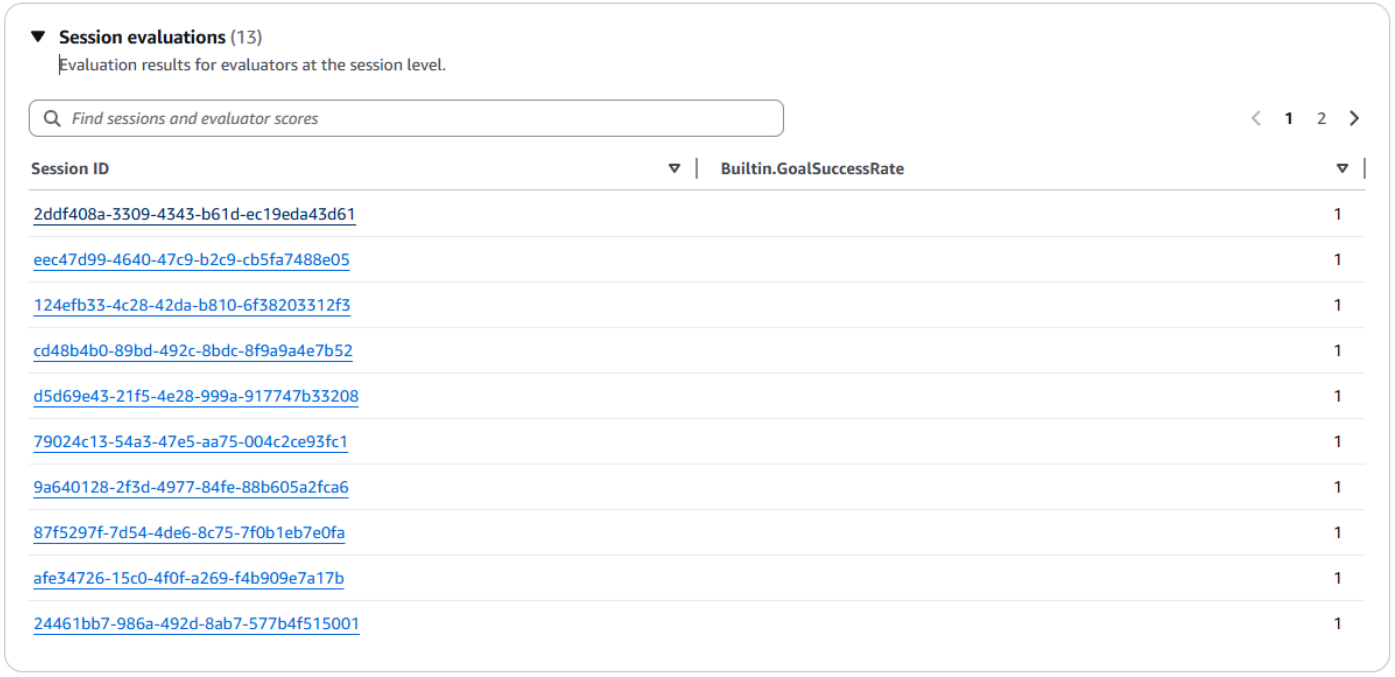
Evaluation configuration metrics

> Provides metrics for the overall evaluation configuration. An evaluator defines how to assess a specific aspect of an AI agent's performance. To view more details about an evaluator, choose its name in the **Evaluator** column. To view a bar chart and analyze trends for an evaluator, choose the value in the **Count** column.



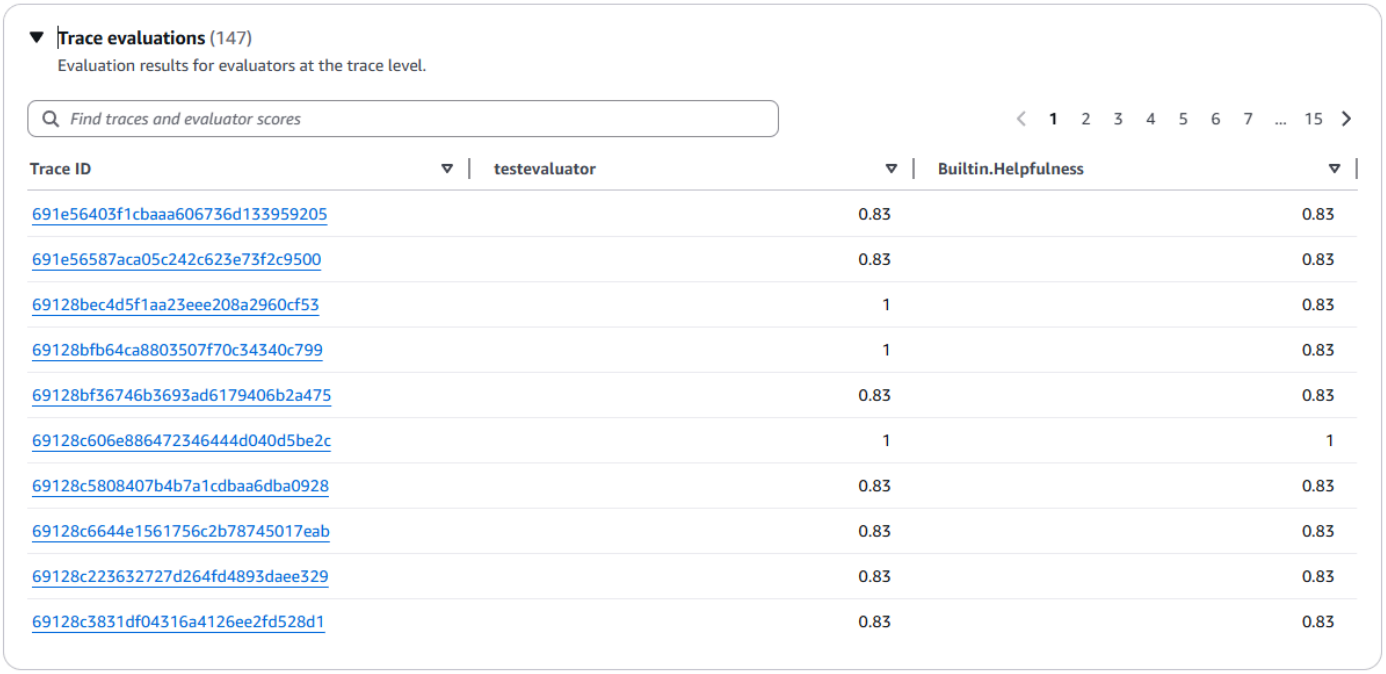
Session evaluations

Provides evaluation results for evaluators at the session level. A session represents a logical grouping of related interactions from a single user or workflow. A session can contain one or more traces. You can choose a session to filter down to the list of traces within that session in the **Trace evaluations** section.



Trace evaluations

Provides evaluation results for evaluators at the trace level. A trace is a complete record of a single agent execution or request. A trace can contain one or more spans. Choose a trace to view the trace details along with all the evaluators that were run on that trace.



Trace evaluations (147)

Evaluation results for evaluators at the trace level.

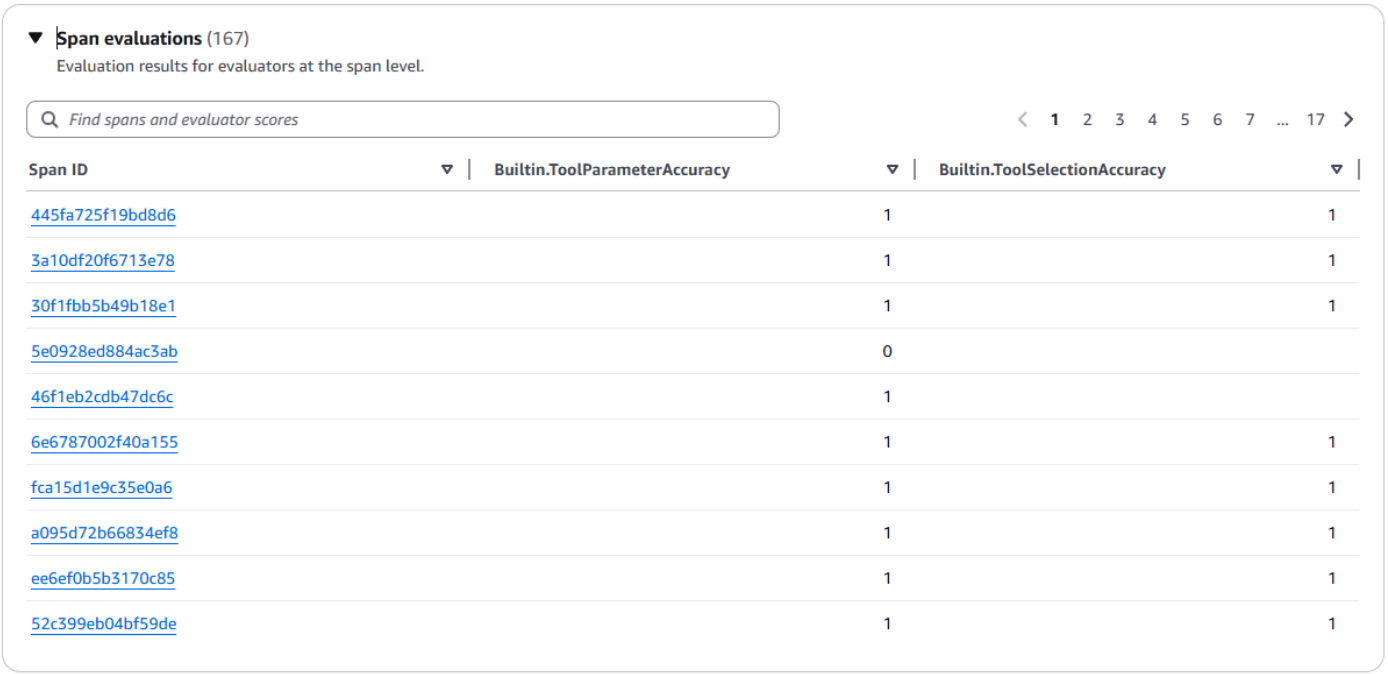
Find traces and evaluator scores

1 2 3 4 5 6 7 ... 15

Trace ID	testevaluator	Builtin.Helpfulness
691e56403f1cbaaa606736d133959205	0.83	0.83
691e56587aca05c242c623e73f2c9500	0.83	0.83
69128bec4d5f1aa23eee208a2960cf53	1	0.83
69128bfb64ca8803507f70c34340c799	1	0.83
69128bf36746b3693ad6179406b2a475	0.83	0.83
69128c606e886472346444d040d5be2c	1	1
69128c5808407b4b7a1cdbaa6dba0928	0.83	0.83
69128c6644e1561756c2b78745017eab	0.83	0.83
69128c223632727d264fd4893daee329	0.83	0.83
69128c3831df04316a4126ee2fd528d1	0.83	0.83

Span evaluations

Provides evaluation results for evaluators at the span level. A span represents the individual operations performed during that execution. Choose a span to view the span details along with all the operations performed during that span.



Span evaluations (167)

Evaluation results for evaluators at the span level.

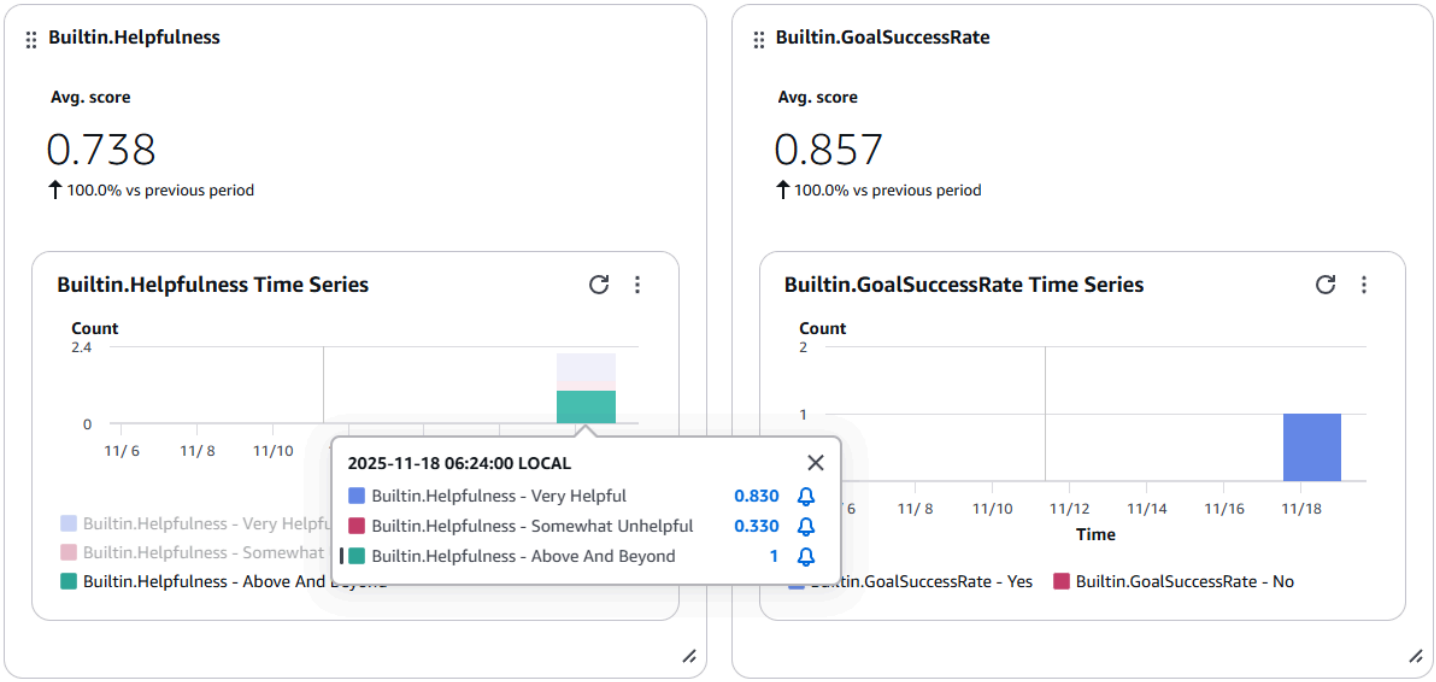
Find spans and evaluator scores

1 2 3 4 5 6 7 ... 17

Span ID	Builtin.ToolParameterAccuracy	Builtin.ToolSelectionAccuracy
445fa725f19bd8d6	1	1
3a10df20f6713e78	1	1
30f1fbb5b49b18e1	1	1
5e0928ed884ac3ab	0	
46f1eb2cdb47dc6c	1	
6e6787002f40a155	1	1
fca15d1e9c35e0a6	1	1
a095d72b66834ef8	1	1
ee6ef0b5b3170c85	1	1
52c399eb04bf59de	1	1

Evaluations graphs

The Evaluations dashboard also includes a bar graph for each evaluator. The graphs show the trends for each evaluator over time, and enable you to set alarms for specific metric values. To set an alarm, click a bar in the graph, and then choose **Alarm** (bell) icon. For more information, see Using Amazon CloudWatch alarms.



Work with evaluation results

If you need direct access to your evaluation results data, or if you want to create custom visualizations or work outside the AgentCore Evaluations console, you can access your evaluation results directly through CloudWatch Logs, CloudWatch Metrics, and CloudWatch dashboards.

Topics

- Accessing evaluation results in CloudWatch Logs
- Accessing evaluation metrics in CloudWatch Metrics
- Creating Custom Dashboards
- Setting alarms on evaluation metrics
- Additional Resources

Accessing evaluation results in CloudWatch Logs

Your evaluation results are automatically published to CloudWatch Logs in Embedded Metric Format (EMF).

To find your evaluation results log group

1. Open the CloudWatch console.
2. In the navigation pane, choose **Logs Management** > **Log groups**.
3. Search for or navigate to the log groups with prefix: `/aws/bedrock-agentcore/evaluations/`.
4. Within this log group, the log events contain the evaluation results.

For more information about working with log groups and querying log data, see Working with Log Groups and Log Streams and Analyzing Log Data with CloudWatch Logs Insights.

Accessing evaluation metrics in CloudWatch Metrics

Evaluation results metrics are automatically extracted from the Embedded Metric Format (EMF) logs and published to CloudWatch Metrics.

To find your evaluation metrics

1. Open the CloudWatch console.
2. In the navigation pane, choose **Metrics** > **Classic metrics**.
3. Select the **Bedrock AgentCore/Evaluations** namespace.
4. Browse available metrics by dimensions.

For more information about viewing and working with metrics, see Using CloudWatch Metrics and Graphing Metrics.

Creating Custom Dashboards

You can create custom dashboards to visualize your evaluation metrics alongside other operational metrics.

To create a dashboard with evaluation metrics

1. In the CloudWatch console, choose **Dashboards** from the navigation pane.

2. Choose **Create dashboard**.
3. Add widgets and select metrics from the **Bedrock AgentCore/Evaluations** namespace.
4. Customize the time range, statistic, and visualization type for your needs.

For detailed instructions, see Creating and Working with Custom Dashboards and Using CloudWatch Dashboards.

Setting alarms on evaluation metrics

You can set alarms to notify you when evaluation metrics cross specified thresholds that you have specified, such as when correctness drops below acceptable levels.

To create an alarm on evaluation metrics

1. In the CloudWatch console, choose **Alarms** > **All alarms**.
2. Choose **Create alarm**.
3. Choose **Select metric** and navigate to the **Bedrock AgentCore/Evaluations** namespace.
4. Select the metric you want to monitor.
5. Configure the threshold conditions (dynamic anomaly detection threshold available where you don't need to specified a static number threshold) and notification actions.

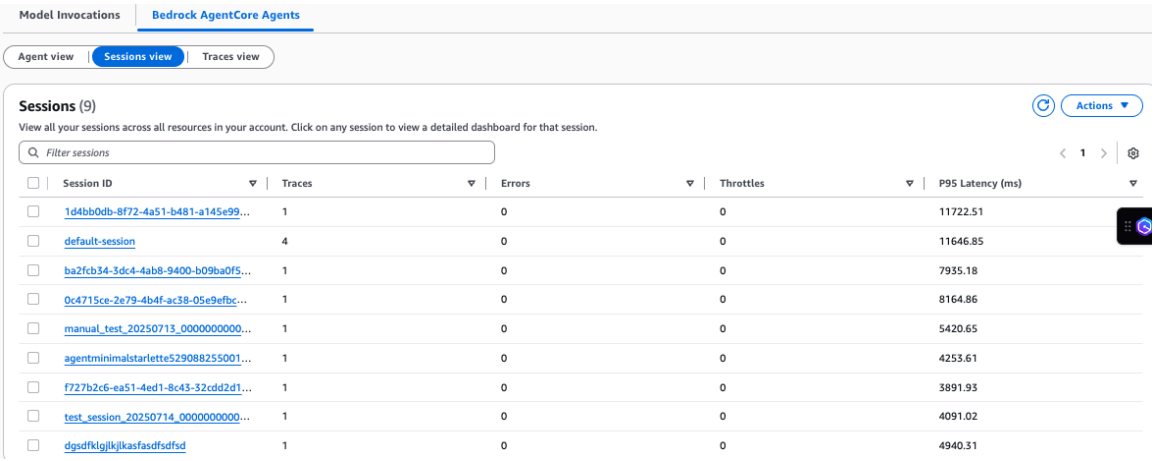
For detailed instructions, see Using CloudWatch Alarms and Creating a CloudWatch Alarm Based on a Static Threshold.

Additional Resources

- CloudWatch Embedded Metric Format
- CloudWatch Logs Insights Query Syntax
- Creating Composite Alarms

Session view

The **Sessions** view shows the list of all the sessions associated with all agents in your account. Choose **Filters** or sort by columns to find a specific session. Choose a session under **Session ID** to view the session details.



You can view the Session summary metrics and the list of traces belonging to that session. Session metrics include:

- Traces – Number of traces belonging to the sessions
- Server errors – Count of system errors during request processing. High levels of server-side errors can indicate potential infrastructure or service issues that require investigation
- Client errors – Client errors are errors resulting from invalid requests. High levels of client-side errors can indicate issues with request formatting or permissions
- Throttles – Number of requests throttle relevant to this session due to exceeding allowed TPS (Transactions Per Second)
- Sessions details – Meta data about the session such as start time, session ID

> **Note**
>
> Summary page fields are consistent across **Agent view**, **Sessions view**, and **Traces view**. For more information on summary fields, see Agent view.

Under **Traces** for a session, choose **Filter traces** to find the trace you want to review. After you choose a trace, view the trace details in the right-pane. You can view the trace summary, spans, and trace content for the selected trace.

Traces view

Trace view lists all traces from your agents in this account. To work with traces:

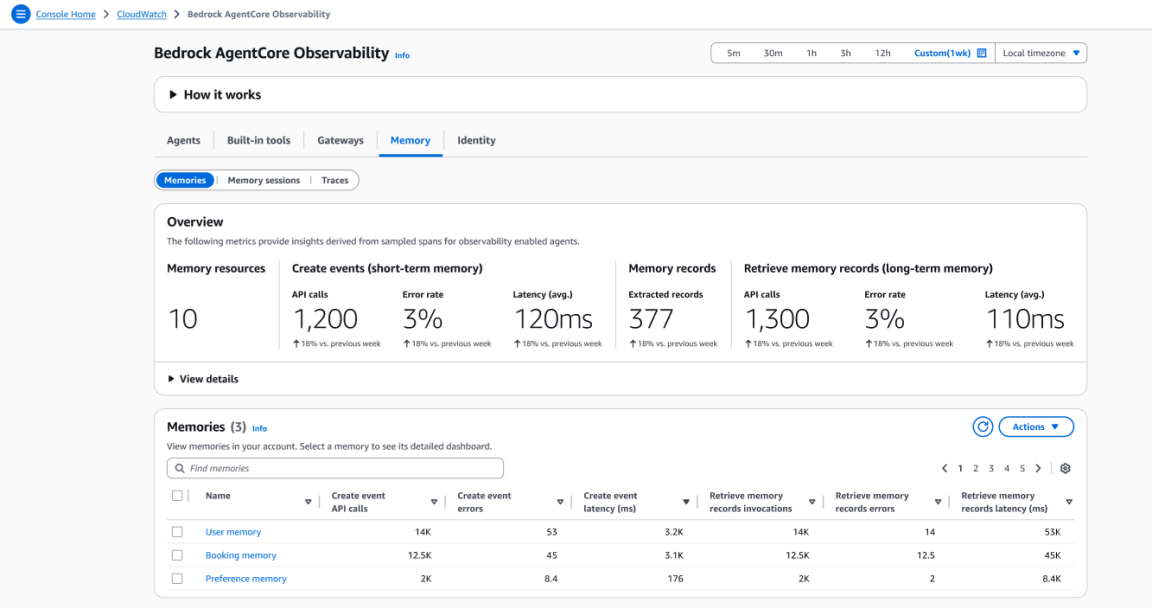
1. Choose **Filter traces** to search for specific traces.
2. Sort by column name to organize results.
3. Under **Actions**, select **Logs Insights** to refine your search by querying across your log and span data or select **Export selected traces** to export.

> **Note**
>
> Summary page fields are consistent across **Agent view**, **Sessions view**, and **Traces view**. For more information on summary fields, see Agent view.

Memory

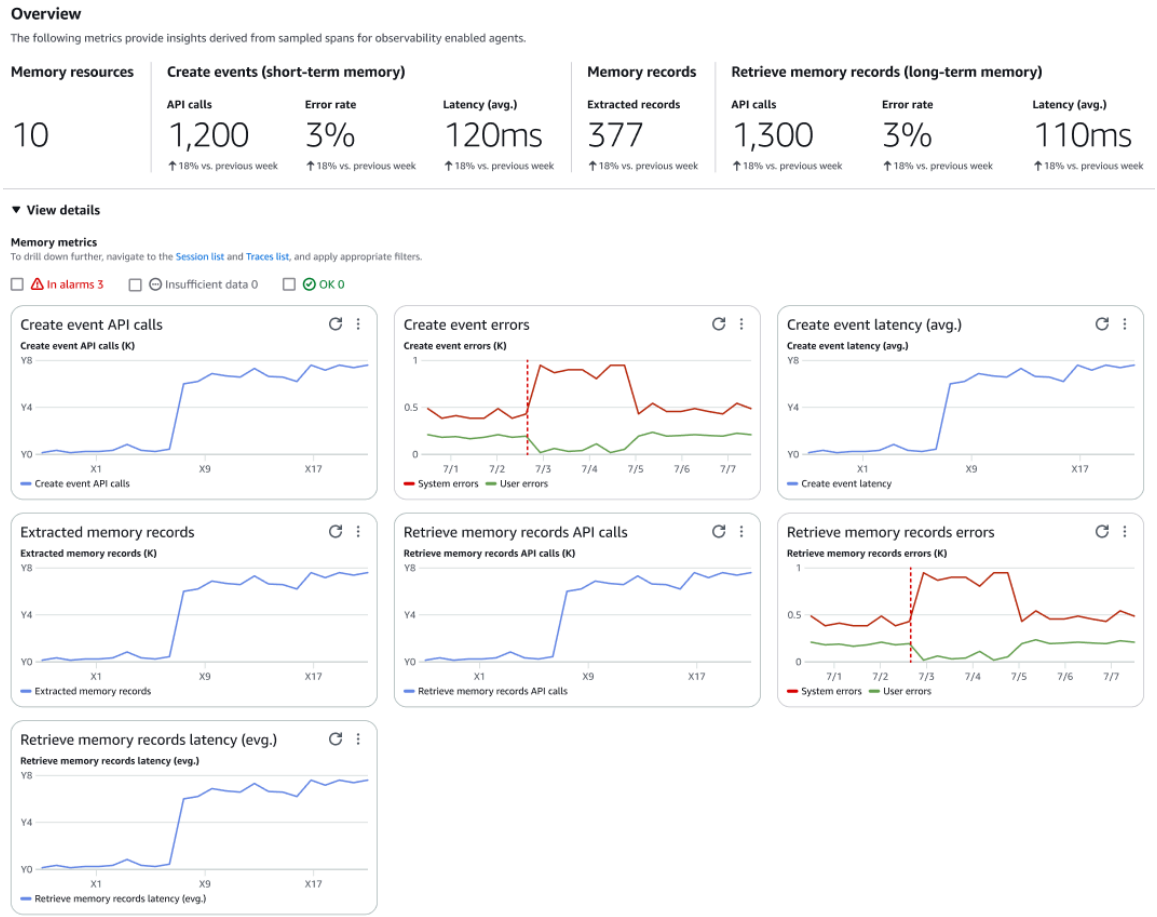
Understand how your agents store, retrieve, and use contextual information to provide personalized experiences. For more information on Amazon Bedrock AgentCore Memory, see Add memory to your AI agent. Memory observability includes three key monitoring areas:

- **Memories** – Monitor memory storage and retrieval patterns
- **Memory sessions** – Monitor memory usage within individual sessions
- **Traces view** – Access detailed trace information for memory operations

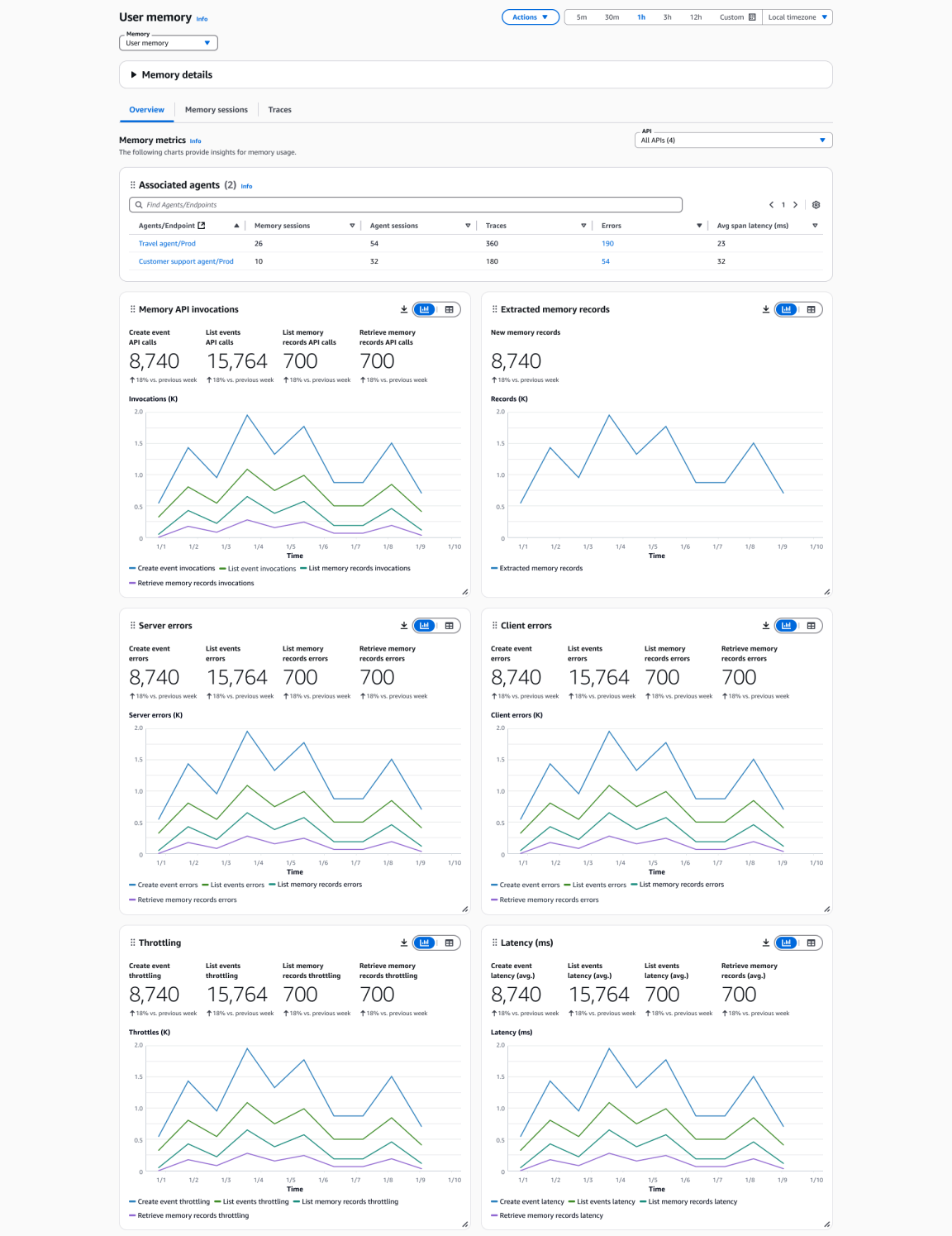


To understand short-term or long-memory, see Add memory to your AI agent .

Choose **View details** to view the memory metrics in graphs.



Under **Memories**, you can view all the memories associated with your account. Choose a memory **Name** to view the memory details.



On the **Memory details** page, you will see the following tabs:

- **Overview** – Displays comprehensive memory performance metrics and usage patterns for the selected memory resource

- **Associated agents** – You can view the agents using the memory. Choose an **Agent/Endpoint** to view the agent overview page.
- **Memory API invocations** – Total number of API calls made to memory operations including storage, retrieval, and update requests. This metric helps track memory system usage and capacity planning
- **Extracted memory records** – Count of memory records successfully extracted and processed from agent interactions. This includes contextual information, user preferences, and conversation history that agents store for personalization
- **Server errors** – Count of system errors during memory operations. High levels indicate potential infrastructure issues with memory storage or retrieval systems that require investigation
- **Client errors** – Errors resulting from invalid memory requests, malformed data, or permission issues. High client error rates may indicate problems with agent memory integration or data formatting
- **Throttling** – Number of memory requests throttled due to exceeding allowed transaction limits. Monitor this metric to determine if memory access patterns need optimization or if service quotas require adjustment
- **Latency** – Response time for memory operations including storage and retrieval requests. Track P50, P90, and P99 latencies to identify performance bottlenecks and optimize memory access patterns

- **Memory sessions** – You can view the session that contain short-term memory from agent interactions. Under **Memory sessions**, choose **Session ID** to view the session dashboard.
- **Traces** – Displays the traces for agents. Under **Traces**, choose **Trace ID** to view the traces that invoke a specific memory. You can use the traces dashboard to deep dive into the agent memory usage, and final responses.

> **Note**
>
> The **Memory sessions** and **Traces** tab experience and fields are similar across **Built-in tools**, **Gateways**, **Memory**, and **Identity** observability. For more information on the fields, see Code interpreter tool.

Built-in tools

Use CloudWatch to gain visibility into how your agents use built-in tools like `Code interpreter tool` and `Browser use tool` to complete tasks. CloudWatch provides monitoring capabilities for each tool. For more information on Amazon Bedrock AgentCore built-in tools, see Use Amazon Bedrock AgentCore built-in tools to interact with your applications .

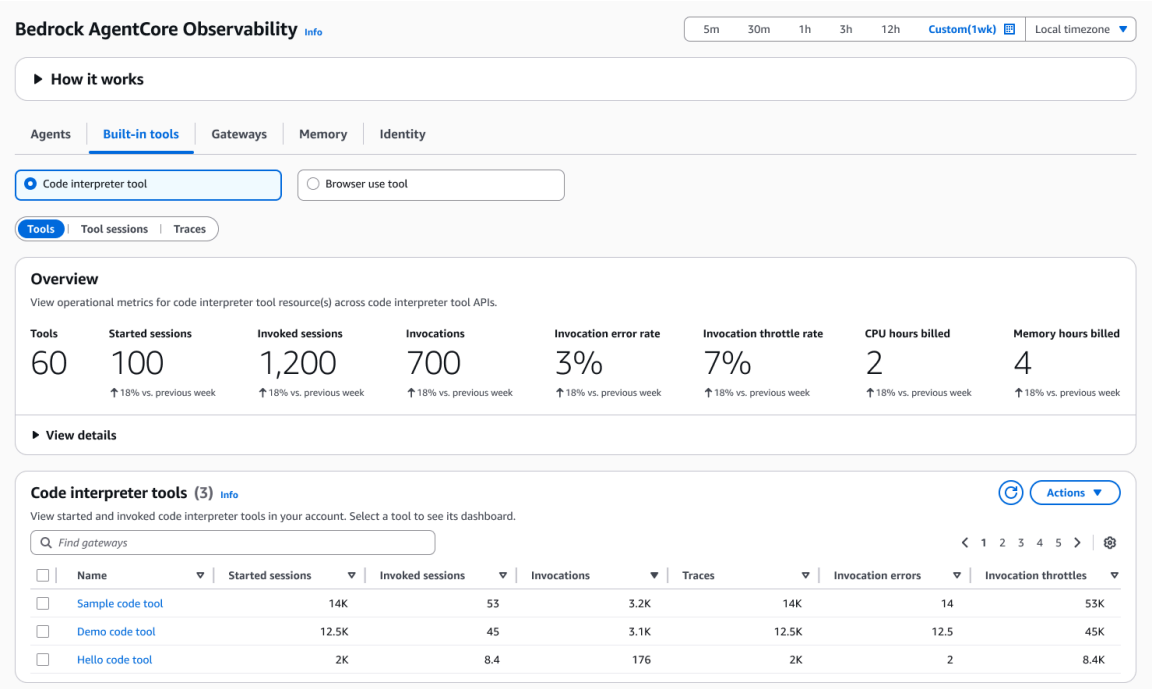
Topics

- Code interpreter tool
- Browser use tool

Code interpreter tool

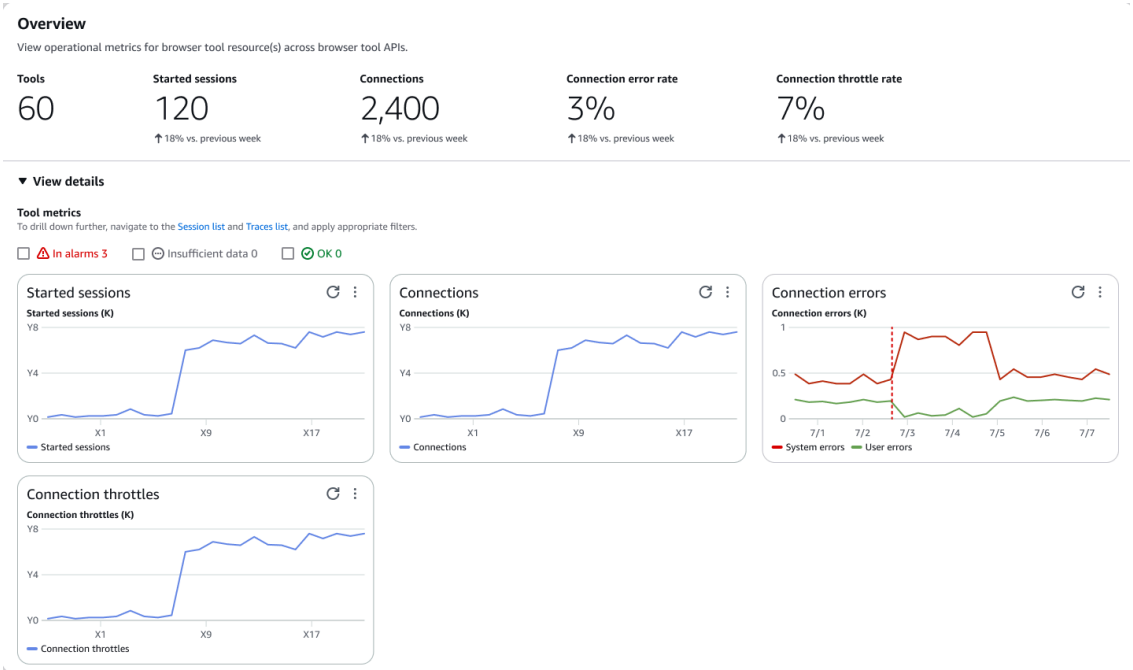
You can use the code interpreter tool for the following:

- Track code execution success rates, runtime duration, and resource consumption
- Monitor memory usage and computational resource allocation
- Analyze code execution patterns and optimization opportunities
- Observe error rates and debugging information for failed executions
- Track security sandbox isolation and compliance metrics



- **Tools** – You can monitor fetch operations and API calls made through the browser tool. Choose a tool under **Name** to view the dashboard.

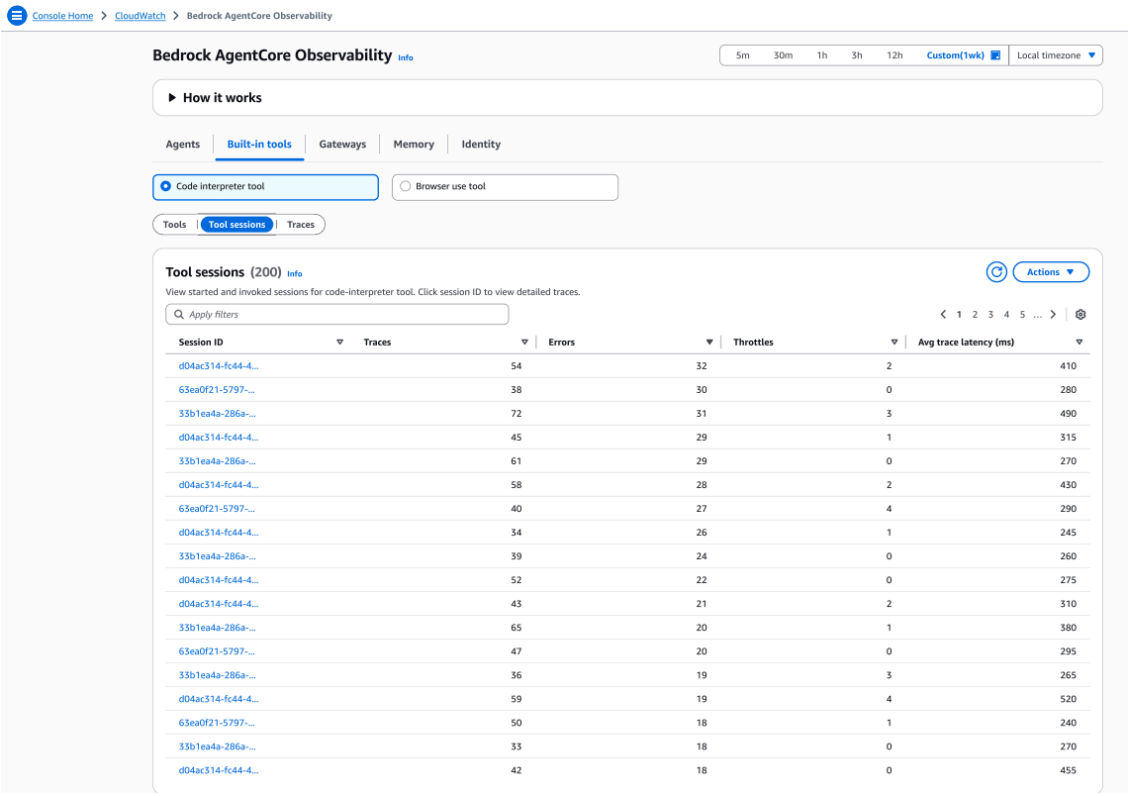
  Choose **View details** to view the resource details.



- **Started sessions** – Total number of code interpreter sessions initiated by agents. Each session represents a sandboxed environment where agents can execute code, analyze data, and generate outputs. Monitor this metric to track code interpreter usage and capacity planning
- **Connections** – Number of active connections to code interpreter runtime environments. This includes both successful connections and connection attempts, helping track resource utilization and concurrent usage patterns
- **Connection errors** – Count of failed connections to code interpreter environments due to system issues, resource constraints, or configuration problems. High connection error rates may indicate infrastructure issues requiring investigation
- **Connection throttles** – Number of connection requests throttled due to exceeding allowed limits or resource constraints. Monitor this metric to identify when code interpreter usage approaches capacity limits and may require scaling
- **CPU hours billed** – Total computational time consumed by code interpreter sessions, measured in CPU hours. This metric helps track resource costs and optimize code execution efficiency across agent workloads

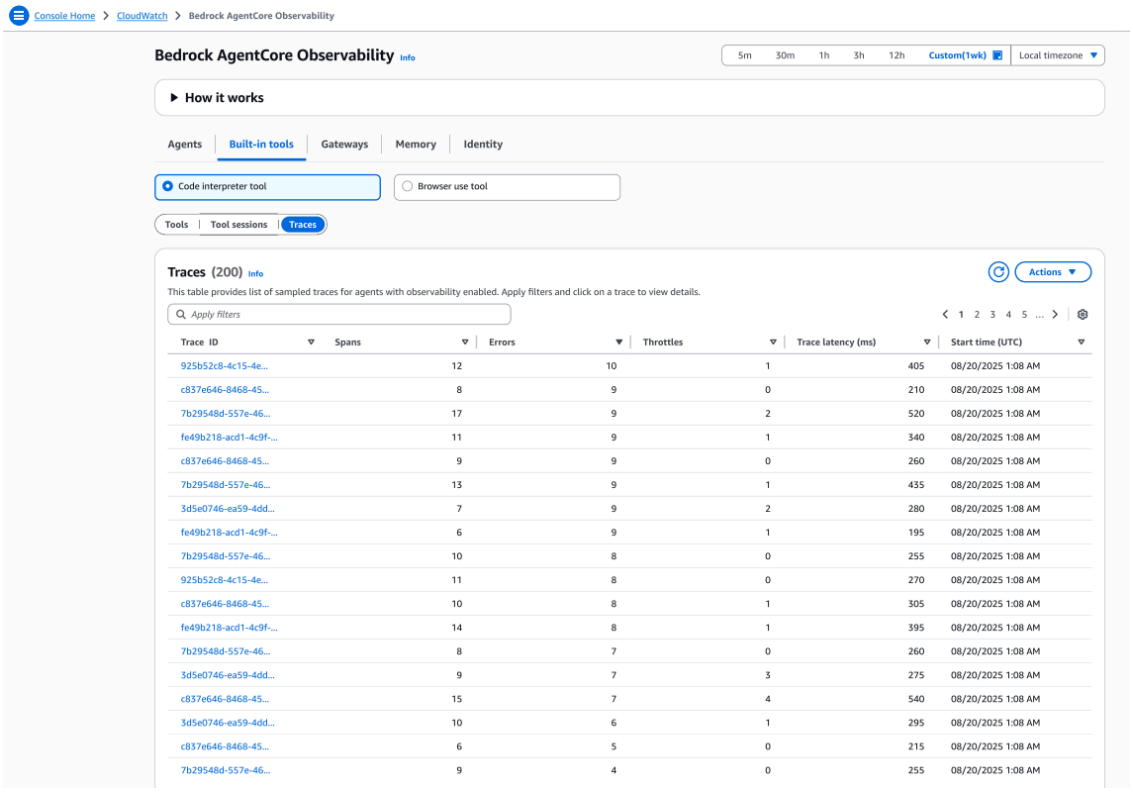
- **Memory hours billed** – Total memory consumption by code interpreter sessions over time, measured in memory hours. Use this metric for cost tracking and to identify memory-intensive code execution patterns that may need optimization

- **Tool sessions** – View all the connected sessions where the tool was used.

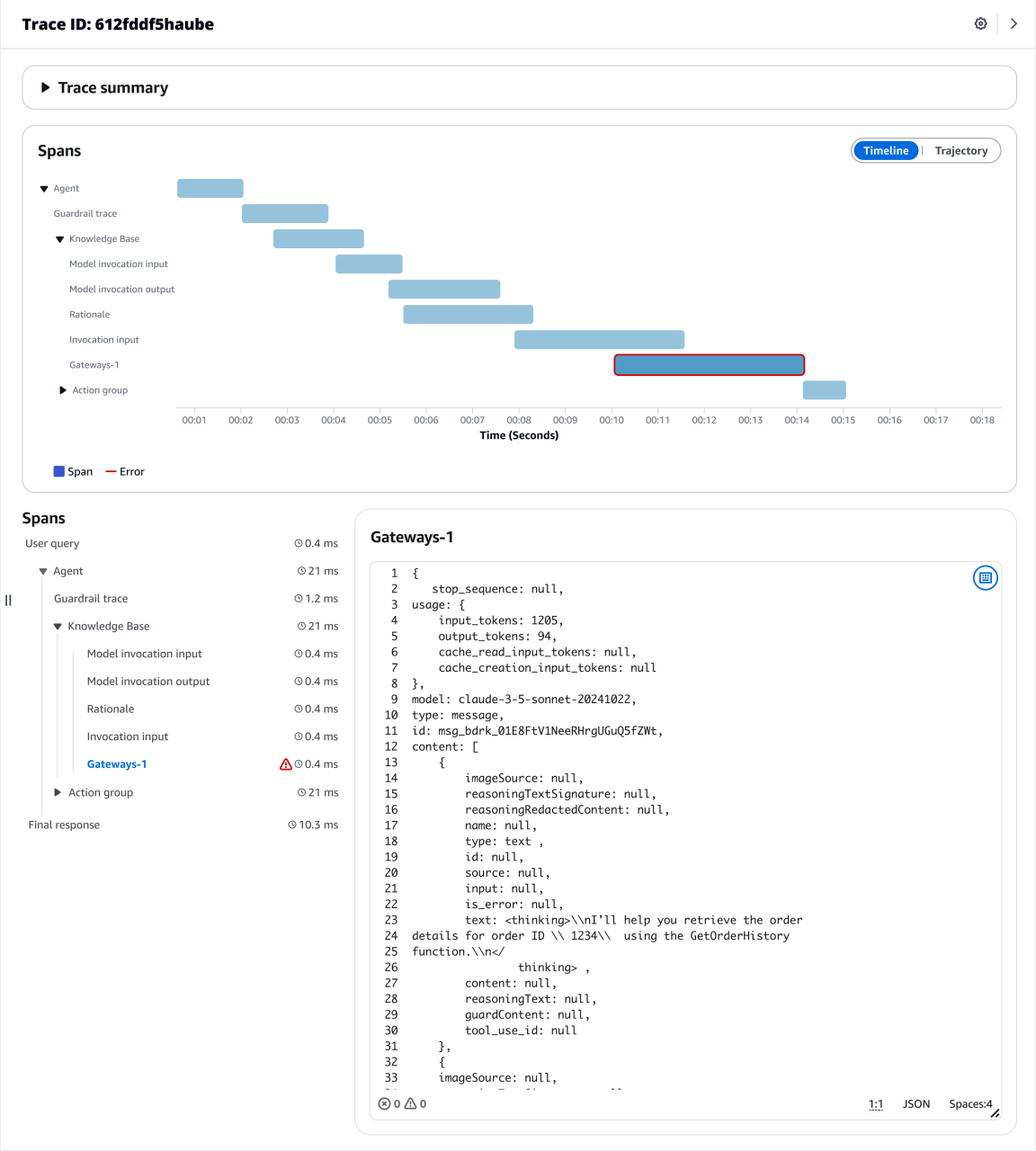


  Choose a **Session ID** under **Total sessions** to view the session dashboard.

- **Traces** – View the sample traces for agents with observability enabled.



Choose a **Trace ID** under **Traces** to view the trace details.



Browser use tool

You can use the browser use tool for the following:

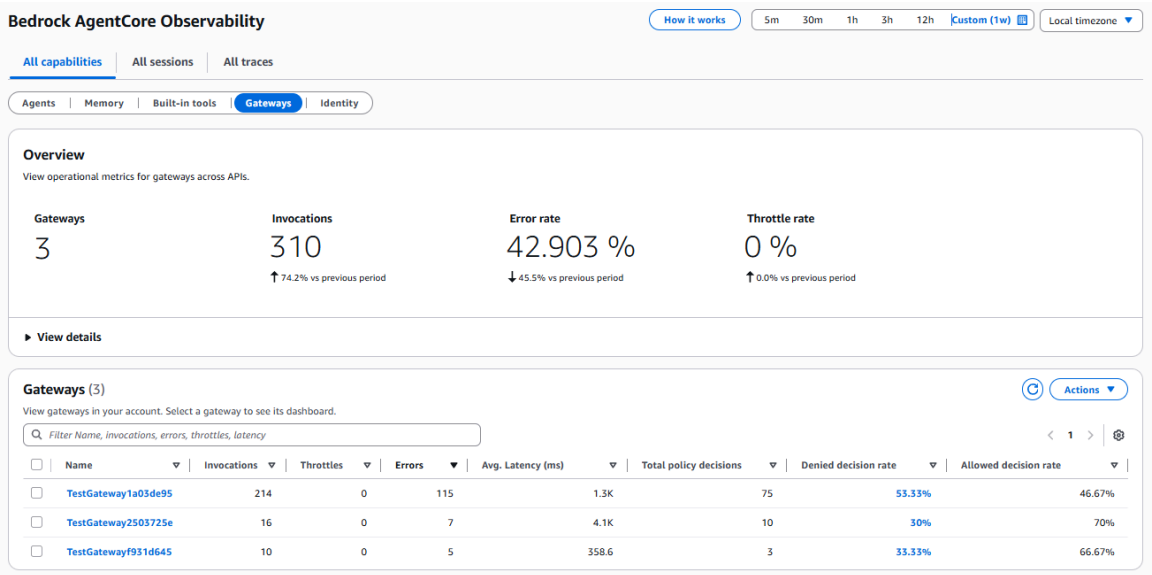
- Track browser navigation patterns, page load times, and interaction success rates
- Observe browser session duration and resource utilization
- Analyze fetch errors and troubleshoot browser automation issues
- Observe browser session duration and resource utilization
- Track security sandbox performance and isolation effectiveness

> **Note**
>
> The **Tools**, **Tool sessions**, and **Traces** tab experience and fields are similar to **Code interpreter tool**. For more information on the fields, see Code interpreter tool.

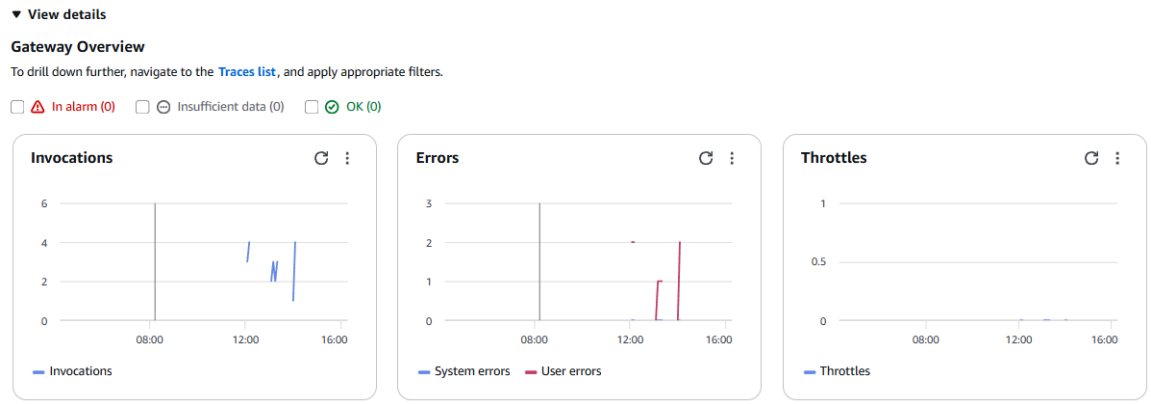
Gateways

Monitor how your agents discover and interact with external tools and services through AgentCore Gateway. For more information on Amazon Bedrock AgentCore Gateway, see Amazon Bedrock AgentCore Gateway. Gateway observability includes comprehensive monitoring across multiple areas:

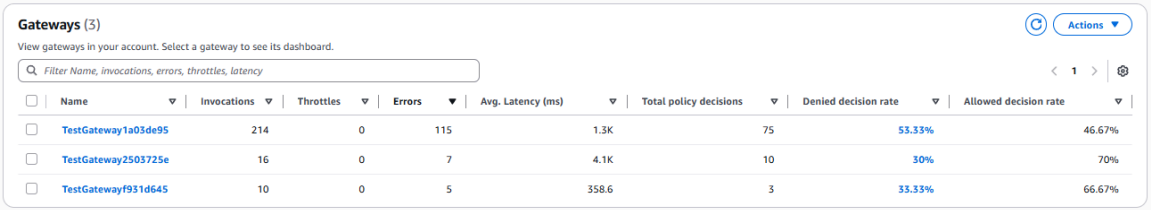
- Track API transformation success rates and response times for external service calls
- Monitor tool discovery patterns and usage frequency across different agents
- Analyze authentication and authorization flows for third-party service access
- Observe data transformation accuracy when converting between different API formats
- Track error rates and retry patterns for external service integrations



Expand the **View details** section to view the gateway metrics in graphs.



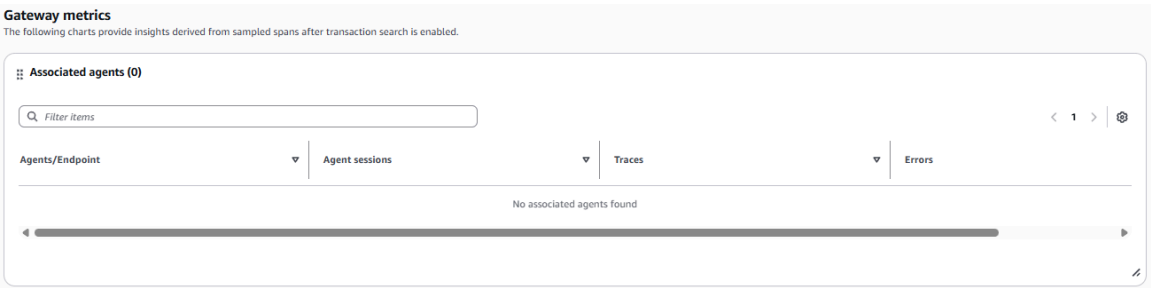
Under **Gateways**, choose a gateway **Name** to view the dashboard. You can also sort the list of gateways by click the column headers in the table.



Gateway details - Overview

The **Overview** tab provides insights derived from sampled spans after transaction search is enabled.

The **Gateway metrics** section lists all of the agents associated with the selected gateway, and it provides information about the number of sessions, traces and errors for each associated agent.



Additionally, the **Overview** tab includes the following interactive charts.

Invocations

The **Invocations** chart shows the total number of API requests made. Each API call counts as one invocation, regardless of the response status.

System and client errors

The **System and client errors** chart provides information about the number of API requests that failed with system erros (`5xx` error codes) and user errors (`4xx` status codes, excluding `429`), and the overall error rate as a percentage.

Throttles

The **Throttles** chart provides information about the number API requests that were throttled (`429` status code) by the service, and the overall throttle rate as a percentage.

Invocation latency

The **Invocation and latency** chart provides information about the average latency for list and call tools. It also provides information about average latency between when the service received the request and when it begins sending the first response token.

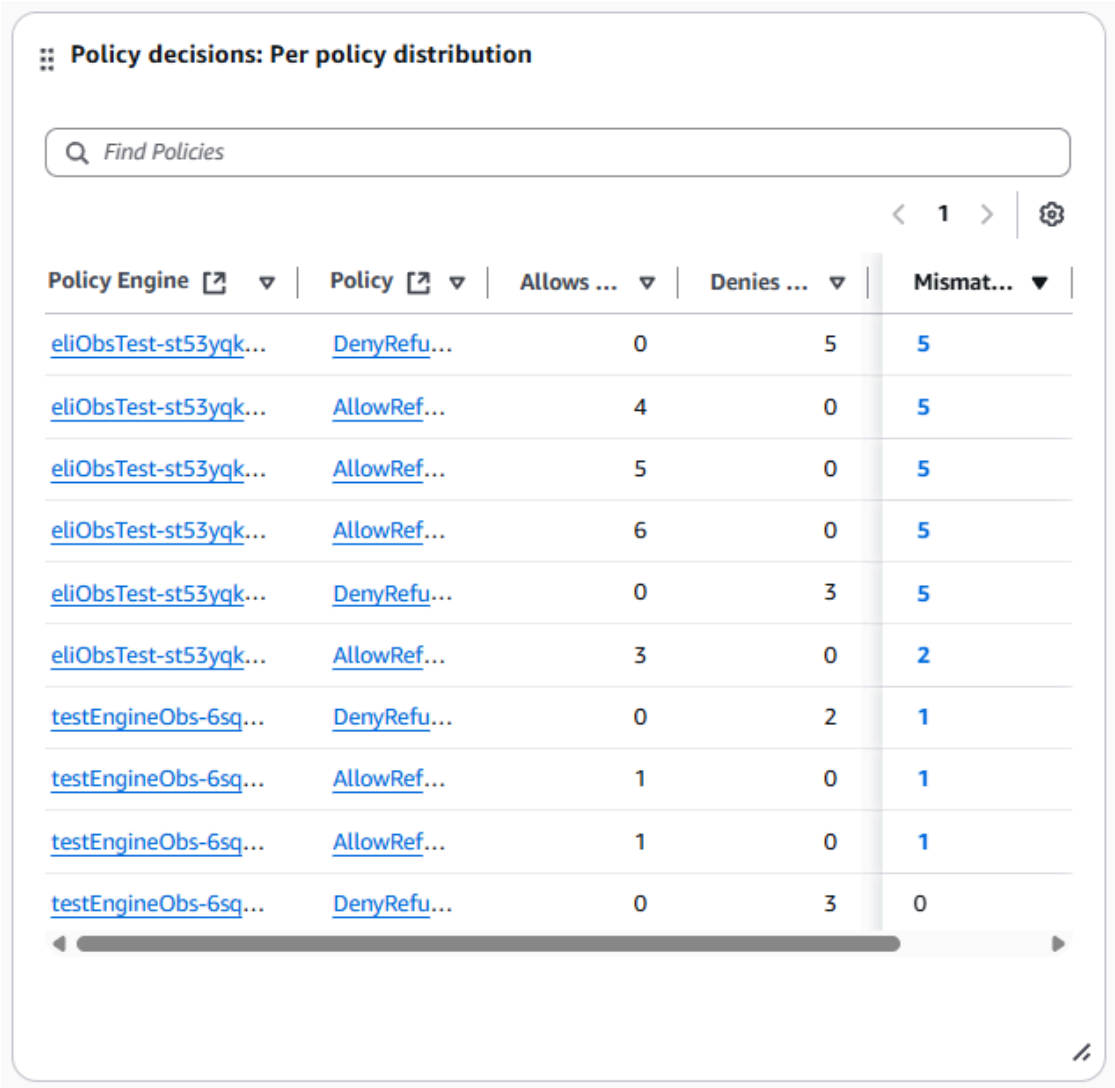
Policy decisions over time

The **Policy decisions over time** chart provides information about the number of decisions that resulted in `allow` and `deny` authorization actions. To view decisions for a specific policy, select the policy in the drop-down.



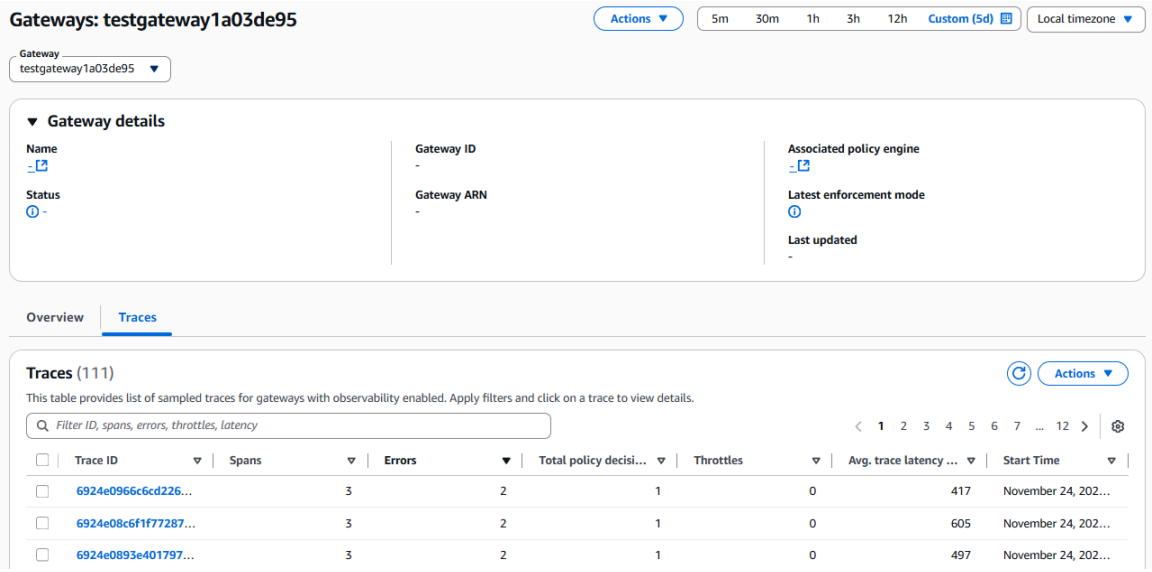
Policy decisions: Per policy distribution

The **Policy decisions: Per policy distribution** chart lists all of the policy engines associated with the selected gateway, and shows the policy, number of allows and denies, and enforcement mode for each policy engine. You can choose a policy engine or policy in the list to view more details about it in the Amazon Bedrock AgentCore console.



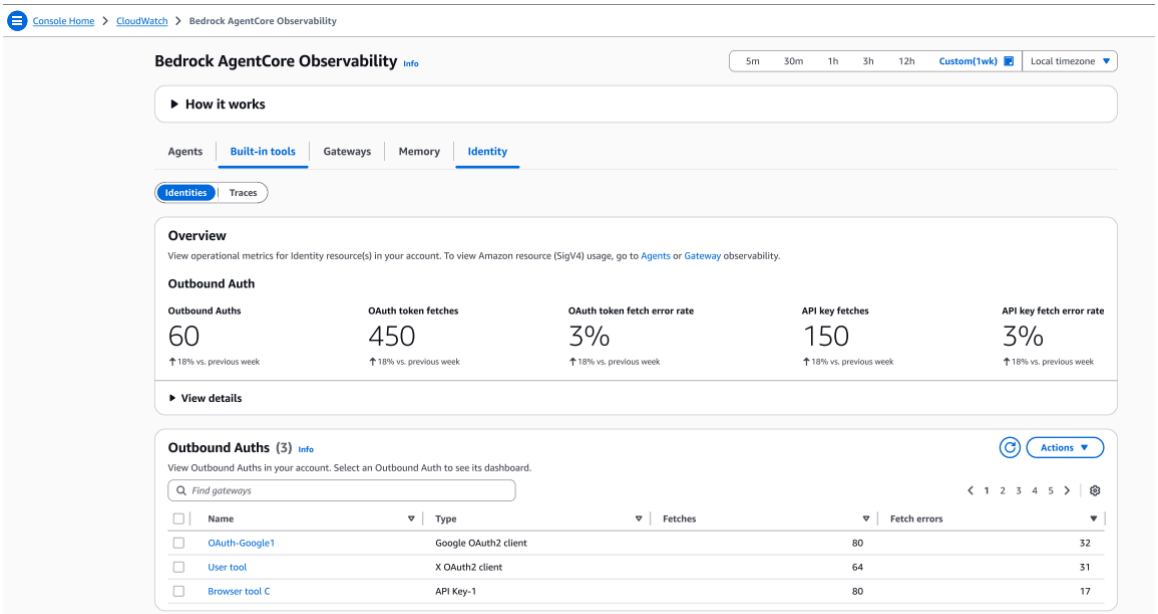
Gateway details - Traces

The **Traces** tab displays all of the sampled traces for the selected gateway. In the **Traces** section, choose a **Trace ID** to view to view metrics for the trace and all of the spans within it.



Identity

Track identity and access management operations to help keep agent behavior secure and compliant. For more information on Amazon Bedrock Identity, see Create agent and tool identities with AgentCore Identity . Identity observability includes monitoring for different authentication methods:

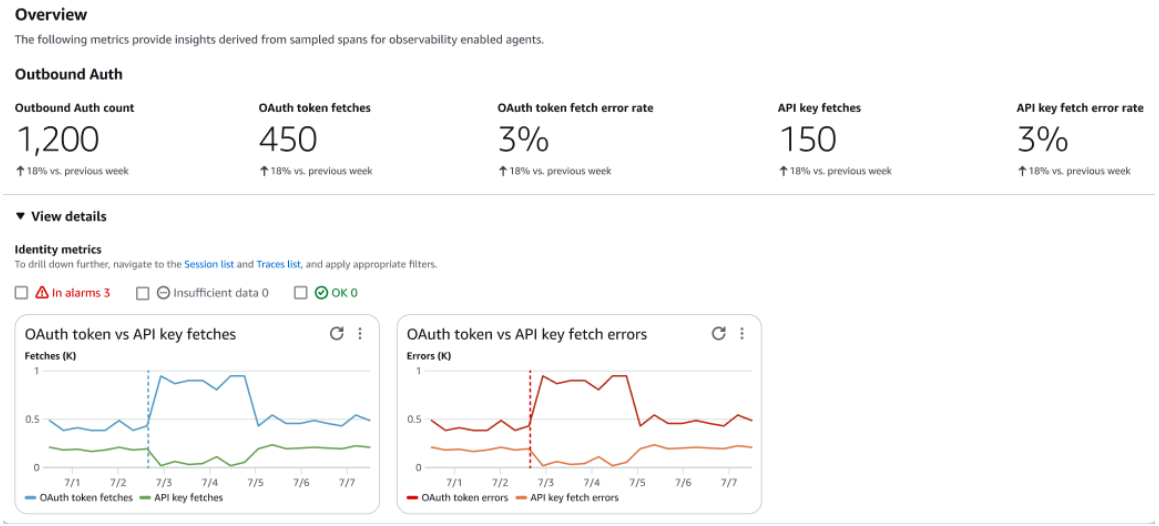


- **Identities** – Access detailed trace information for identity operations
- **Traces** – Apply advanced filters to analyze specific trace patterns

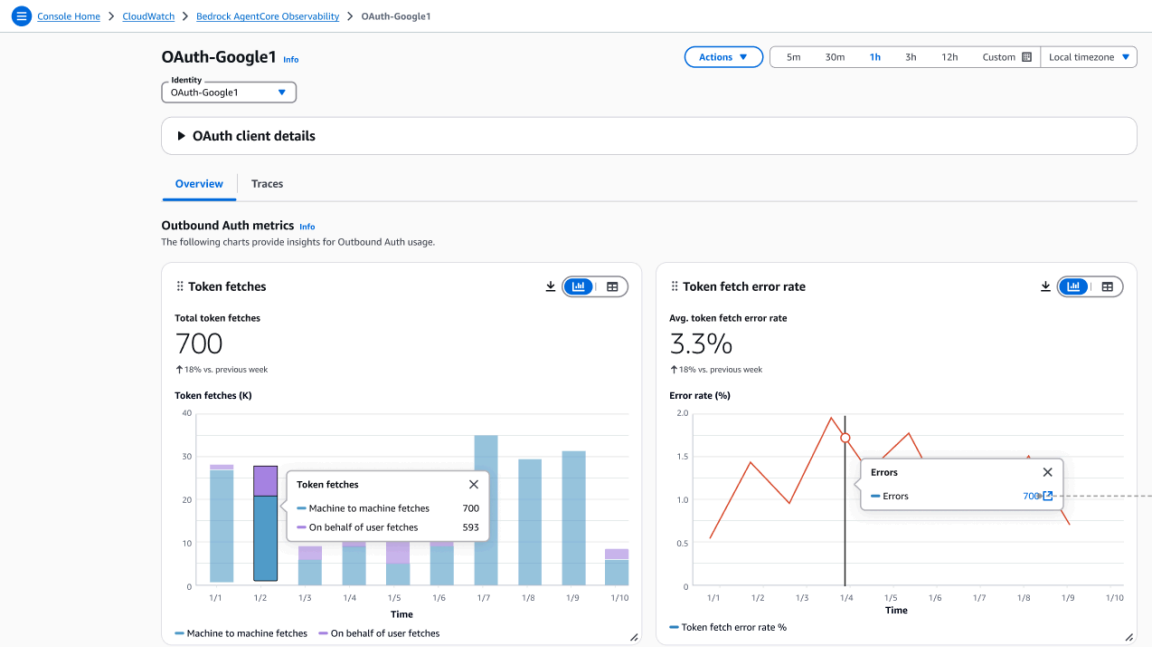
Under **Identities**, you will see the following:

- **Outbound Auth** – Total number of outbound authentication requests initiated by Amazon Bedrock AgentCore to external identity providers
- **OAuth token fetches** – Number of OAuth access tokens successfully retrieved from configured OAuth providers for agent authentication
- **OAuth token fetch error rate** – Percentage of OAuth token retrieval attempts that failed due to network issues, invalid credentials, or provider errors
- **API key fetches** – Number of API keys retrieved from configured key management systems for authenticating agent requests
- **API key fetch error rate** – Percentage of API key retrieval attempts that failed due to access issues, invalid keys, or system errors

Choose **View details** to see the Identity metrics in graphs.

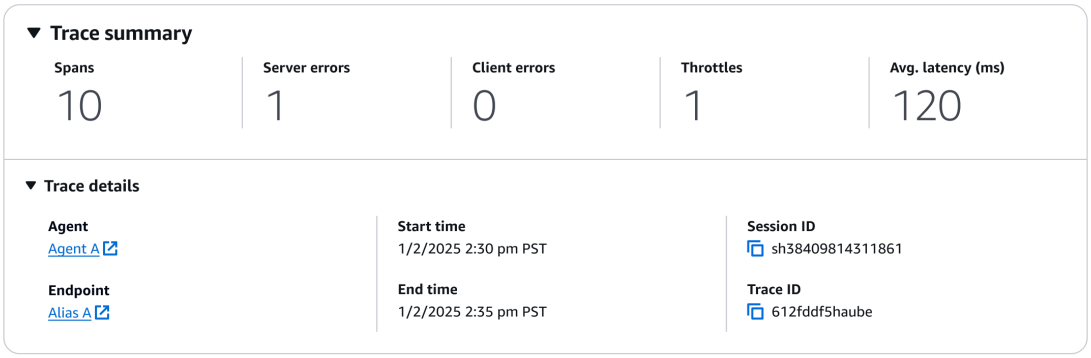


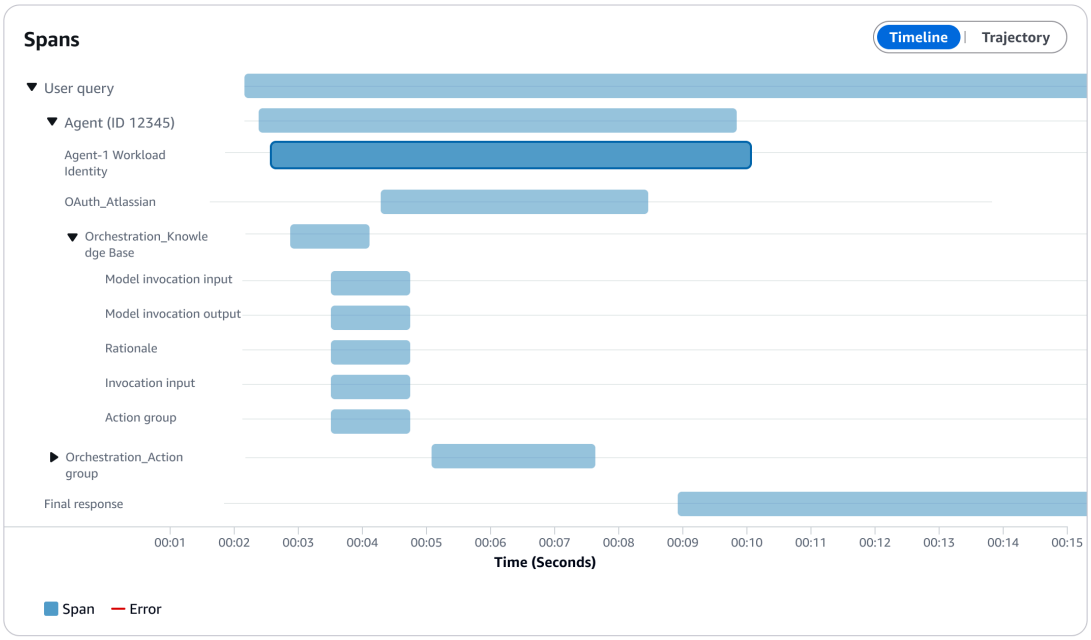
Under **Outbound Auths**, choose a outbound auth **Name** to view the dashboard.



On the **OAuth client details** page, you will see the following tabs:

- **Overview** – Displays comprehensive outbound authentication usage metrics and patterns for OAuth clients
  - **Token fetches** – Total number of authentication token requests made by agents, including both machine-to-machine and on-behalf-of-user authentication flows. This metric tracks overall authentication activity and helps with capacity planning for identity services
  - **Token fetch error rate** – Percentage of failed token requests out of total authentication attempts. Monitor this metric to identify authentication issues, expired credentials, or permission problems. Trends over time help detect degrading authentication performance
- **Traces** – Displays detailed trace information for identity and authentication operations, including OAuth flows, workload identity token requests, and third-party service integrations. Use traces to troubleshoot authentication failures, analyze token fetch latency, and monitor security compliance across agent interactions





Trace steps

- User query — 0.4 ms
- **Agent (ID 12345)** — 21 ms
  - **Agent-1 Workload Identity: Access token** — 0.58 ms
  - OAuth-Google1 — 1.2 ms
  - Orchestration_Knowledge Base — 21 ms
    - Model invocation input — 0.4 ms
    - Model invocation output — 0.4 ms
    - Rationale — 0.4 ms
    - Invocation input — 0.4 ms
    - Action group — 0.4 ms
  - Orchestration_Action group — 21 ms
- Final response — 10.3 ms

Agent-1 Workload Identity: Access token

Span JSON


```
{
  "span_id": "1-5f187253-e8d03c0b64a0aefa24f12b58",
  "name": "GetWorkloadAccessTokenForJWT",
  "start_time": 1595511379.364,
  "end_time": 1595511379.944,
  "http": {
    "request": {
      "method": "POST",
      "url": "/GetWorkloadAccessTokenForJWT"
    },
    "response": {
      "status": 200,
    }
  },
  "request_context": {
    "workload_identity": "Agent-1 Workload Identity",
    "issuer": "https://cognito-idp.us-east-1.amazonaws.com/
us-east-1_RBoqxzC3v",
    "user_sub": "94d81408-d031-7027-1037-9c796d2949eb"
  },
  "response": {
    "payload": {
      "WorkloadAccessToken": "DummyWorkloadAccessToken",
      "expires_in": 3600
    },
    "http_status_code": 200
  },
  "performance_metrics": {
    "request_latency": 580,
    "processing_time": 550,
    "network_time": 30
  },
  "throttling_status": {
```


0 0 1:1 JSON Spaces:4

Monitor AI coding agents with Amazon CloudWatch Coding Agent Insights

AI coding agents such as OpenAI Codex, Claude Code, and GitHub Copilot emit OpenTelemetry (OTel) telemetry about how your developers use them—token consumption, per-turn latency, tool calls, API requests, and approvals. When you send this telemetry to Amazon CloudWatch, you can monitor adoption, attribute usage and cost to teams and departments, and spot performance or reliability issues across your developer fleet.

CloudWatch provides **Coding Agent Insights**, a set of ready-to-use dashboards in the console under **GenAI Observability**, **Coding Agent Insights**. The dashboards let you slice and dice usage across organizational attributes such as organization, department, team, cost center, and user, so you can compare adoption and cost across your fleet. You can also export the underlying data as CSV for further analysis. You don't create or import these dashboards. They appear automatically and populate as soon as your agents send metrics in the expected shape. Your only task is to configure each agent to publish its OTel metrics to CloudWatch.

> **Important**
>
> The Coding Agent Insights dashboards rely on a specific metric and attribute shape. Each agent's native OTel metrics carry the dashboard's core dimensions (such as `model` and `token_type`). Identity and organizational attributes (such as `user.email`, `team.id`, `department`, `cost_center`, and `organization`) must be sent as OTel *resource* attributes. The setup guides in this section configure each agent to emit telemetry in this shape. If you change the metric names or move identity attributes off the resource, the dashboards might not populate.

Prerequisites

Coding agents send metrics to the CloudWatch native OTLP metrics endpoint. Before you configure an agent, become familiar with the endpoint. For more information, see OTLP Endpoints and Getting started.

Setup guides

Choose the guide for your coding agent. Each guide covers one or more setup paths, so you can pick the approach that matches how your developers run the agent.

- Claude Code — Configure Claude Code using a bearer token (individuals and small teams) or an enterprise rollout with corporate SSO.
- Set up OpenTelemetry for OpenAI Codex — Configure OpenAI Codex using a bearer token (individuals and small teams) or an enterprise rollout with corporate single sign-on (SSO).
- Set up OpenTelemetry for GitHub Copilot — Configure GitHub Copilot using a bearer token.

View the dashboards

After an agent starts sending metrics, view the dashboards in the console.

To view the Coding Agent Insights dashboards

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **GenAI Observability**, and then choose **Coding Agent Insights**.
3. Choose the tab for your coding agent. Metrics typically appear within a few minutes of the agent sending data.

> **Note**
>
> Metrics are queryable with PromQL, so you can also build your own dashboards and alarms. For more information, see PromQL querying.

Set up Claude Code OpenTelemetry metrics for Amazon CloudWatch

Claude Code emits OpenTelemetry (OTel) metrics about token usage, sessions, code edits, tool calls, and cost. When you send these metrics to Amazon CloudWatch, they populate the Claude Code tab of the Coding Agent Insights dashboards. For more information about the dashboards, see Coding Agent Insights.

Claude Code configures its OTel exporter through environment variables. Two setup paths are available, depending on how your developers run Claude Code.

Choose a setup path

- **Bearer token** — Claude Code sends OTLP metrics directly to the CloudWatch native OTLP endpoint, authenticating with an `Authorization: Bearer` header. Identity and organizational

attributes are supplied through the `OTEL_RESOURCE_ATTRIBUTES` environment variable. This path needs no collector and is the fastest way for an individual developer or a small team to get started. See Bearer token.

- **Enterprise rollout with Claude apps gateway** — For organizations that need centralized control over developer access, identity federation, and telemetry collection. Deploy a self-hosted gateway that sits between your developers' Claude Code clients and your model provider. Developers sign in with your corporate IdP instead of holding API keys or cloud credentials. The gateway holds the upstream credential, enforces model access and managed settings by IdP group, and relays usage telemetry to your observability stack. Supports Amazon Bedrock, Claude Platform on AWS, Google Cloud, and Microsoft Foundry as upstreams. See Set up Claude Code with the Claude apps gateway.

> **Note**
>
> The bearer token path configures only how Claude Code sends *telemetry* to CloudWatch; it is independent of how Claude Code calls a model for *inference*, so you can use it with any model provider. The enterprise rollout with Claude apps gateway combines telemetry configuration with centralized inference access and supports multiple backends including Amazon Bedrock, Claude Platform on AWS, Google Cloud, and Microsoft Foundry.

For the complete, continually updated guidance, including deployment templates and setup scripts, see the Guidance for Claude Code with Amazon Bedrock on GitHub.

Topics

- Set up Claude Code metrics in Amazon CloudWatch with a bearer token
- Set up Claude Code with the Claude apps gateway

Set up Claude Code metrics in Amazon CloudWatch with a bearer token

On this path, Claude Code sends OpenTelemetry (OTel) metrics directly to the Amazon CloudWatch native OTLP metrics endpoint and authenticates with a bearer token (a CloudWatch metrics API key). No OpenTelemetry collector is required. This is the fastest way for an individual developer or a small team to get Claude Code metrics into the Coding Agent Insights dashboards.

Important

Bearer tokens are long-term credentials. We recommend them only where short-term AWS credentials are not feasible, such as developer laptops outside AWS. For the security trade-offs and key rotation guidance, see Setting up bearer token authentication for Metrics. Organizations that federate identity through corporate SSO should use the enterprise rollout with Claude apps gateway instead. See Set up Claude Code with the Claude apps gateway.

Prerequisites

- Claude Code installed on the developer's machine.
- A working way for Claude Code to call a model for inference. This telemetry setup is independent of inference, so you can use either an Anthropic account (the default Claude Code provider) or Amazon Bedrock. To use Amazon Bedrock for inference, see Claude Code on Amazon Bedrock in the Claude Code documentation.
- A CloudWatch metrics API key (bearer token), created from an IAM user that has CloudWatch metrics permissions. For the procedure that creates the IAM user and generates the key, see Setting up bearer token authentication for Metrics.

Configure the metrics exporter

Claude Code configures OpenTelemetry through environment variables. Set the following variables in the shell that launches Claude Code (or in your shell profile) to enable metrics, select the OTLP/HTTP protocol, and point the metrics exporter at the CloudWatch metrics endpoint with your bearer token in the `Authorization` header. Include the full `/v1/metrics` path.

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_METRICS_ENDPOINT="https://monitoring.region.amazonaws.com/v1/
metrics"
export OTEL_EXPORTER_OTLP_METRICS_HEADERS="Authorization=Bearer YOUR_API_KEY"
```

> **Note**
>
> Treat the bearer token as a secret. Avoid committing it to source control or shell profiles that are shared or backed up insecurely. Rotate the key periodically as described in Setting up bearer token authentication for Metrics.

Add identity and organizational attributes

The Coding Agent Insights dashboards group usage by developer, team, department, and other organizational dimensions. Provide these as OTel *resource* attributes through the `OTEL_RESOURCE_ATTRIBUTES` environment variable. Set the values from your developer's identity and your organization's structure.

```
export OTEL_RESOURCE_ATTRIBUTES="\
user.id=alias,\
user.email=developer@example.com,\
user.name=Jane Doe,\
department=engineering,\
team.id=platform,\
cost_center=cc-1234,\
organization=example-corp,\
location=us-seattle,\
role=software-engineer,\
manager=manager@example.com"
```

> **Note**
>
> Setting these as resource attributes (rather than per-datapoint attributes) matches the shape the dashboards expect, so you can group by `@resource.team.id`, `@resource.department`, and similar labels. Claude Code stamps metric-level dimensions such as `model` and `type` on the metrics automatically.

Verify and view

Run Claude Code to generate metrics, and then confirm the data reached CloudWatch.

To verify metrics

1. Run a Claude Code session that performs some work.
2. In the CloudWatch console, choose **GenAI Observability**, **Coding Agent Insights**, and then the Claude Code tab. Metrics typically appear within a few minutes.
3. Optionally, query the metrics directly with PromQL. For more information, see PromQL querying.

For the complete, continually updated guidance, see the Guidance for Claude Code with Amazon Bedrock on GitHub.

Set up Claude Code with the Claude apps gateway

The Claude apps gateway is a self-hosted service that sits between your developers' Claude Code clients and the model provider. Developers sign in with your corporate identity provider (IdP) instead of holding API keys or cloud credentials. The gateway holds the upstream credential, enforces model access and managed settings by IdP group, and relays usage telemetry to your observability stack—including Amazon CloudWatch.

The gateway is included in the `claude` binary. The same executable that runs Claude Code runs the gateway when you pass the `gateway` subcommand with a configuration file.

Key capabilities

The Claude apps gateway provides centralized control across several dimensions.

- **Credentials** — The upstream API key lives only in your infrastructure. Developers authenticate through SSO with short-lived tokens.
- **Access control** — IdP groups map to model allowlists and managed settings policies.
- **Settings delivery** — The gateway delivers managed settings to signed-in clients.
- **Telemetry** — Routes OTLP metrics (token counts, model, user identity, latency) to your collector. You can fan out to multiple destinations including CloudWatch, Datadog, Splunk, and ClickHouse.
- **Upstream routing** — Clients speak the Anthropic Messages API. The gateway translates for each upstream with failover. Supported upstreams include Amazon Bedrock, Claude Platform on AWS, Google Cloud's Agent Platform, Microsoft Foundry, and the Anthropic API.

Prerequisites

Before you deploy the gateway, confirm that the following requirements are met.

Requirement	Details
Claude Code version	v2.1.195 or later
Identity provider	OpenID Connect (OIDC) provider such as Okta, Microsoft Entra ID, Google Workspace, Keycloak, or Dex
Database	PostgreSQL 14 or later
Model upstream	Amazon Bedrock credentials, Claude Platform on AWS, Google Cloud, Microsoft Foundry, or Anthropic API key
TLS	HTTPS certificate or TLS-terminating ingress
Network	Private-network address (RFC 1918, link-local, CGNAT, IPv6 ULA, or loopback)
Runtime	Linux for production (macOS for local development only)

Deploy the gateway

The following procedure summarizes the quickstart steps. For detailed instructions, see the Claude apps gateway documentation.

To deploy the Claude apps gateway

1. Register an OAuth client in your IdP. Set the redirect URI to `https://claude-gateway.your-domain/oauth/callback`.
2. Provision a PostgreSQL database (any PostgreSQL 14 or later instance). The gateway runs schema migrations at boot.
3. Write a `gateway.yaml` configuration file. The following example shows a minimal configuration that uses Amazon Bedrock as the upstream.

```
listen:
  host: 0.0.0.0
```

```
  port: 8080
  public_url: https://claude-gateway.internal.example.com

oidc:
  issuer: https://login.example.com
  client_id: 0oa1example2
  client_secret: ${OIDC_CLIENT_SECRET}
  allowed_email_domains: [example.com]
  userinfo_fallback: true

session:
  jwt_secret: ${GATEWAY_JWT_SECRET}
  ttl_hours: 1

store:
  postgres_url: ${GATEWAY_POSTGRES_URL}

upstreams:
  - provider: bedrock
    region: us-east-1
    auth: {}

auto_include_builtin_models: true
```

4. Run the gateway. You can use Docker Compose or run the binary directly.

```
claude gateway --config gateway.yaml
```

5. Verify the authentication surface. Fetch the discovery document, request device authorization, and test browser sign-in at your gateway URL.
6. Sign in a developer to confirm end-to-end connectivity.

Connect developers to the gateway

To point developer machines to your gateway, distribute the following managed settings keys through mobile device management (MDM) or your configuration management tool.

```
{
  "forceLoginMethod": "gateway",
  "forceLoginGatewayUrl": "https://claude-gateway.internal.example.com"
}
```

After you distribute the settings, each developer completes the following steps.

To sign in through the gateway

1. Run the `/login` command in Claude Code.
2. Press Enter on the Cloud gateway screen.
3. Complete browser sign-in with your corporate IdP.

After sign-in, Claude Code routes inference through the gateway and the gateway exports telemetry to the configured destinations.

Route telemetry to Amazon CloudWatch

The gateway supports OTLP/HTTP fan-out to multiple collectors, with identity-stamped exports. Configure telemetry destinations in the `gateway.yaml` file. Each destination can receive metrics, logs, or traces.

When you configure CloudWatch as a telemetry destination, the gateway stamps each export with the developer's identity from the IdP. Metrics arrive with user, team, and organizational attributes, so the same Coding Agent Insights dashboards populate.

For the complete telemetry configuration options, see the configuration reference.

Availability and limitations

The following table summarizes what the gateway supports and current limitations.

Item	Status
Supported upstreams	Amazon Bedrock, Claude Platform on AWS, Google Cloud's Agent Platform, Microsoft Foundry, Anthropic API
Supported IdPs	Okta, Microsoft Entra ID, Google Workspace, Keycloak, Dex, and other OIDC-compliant providers
Telemetry protocol	OTLP/HTTP with fan-out to multiple destinations
Runtime	Linux for production deployments. macOS is supported for local development only.

Item	Status
Minimum Claude Code version	v2.1.195

Next steps

Use the following resources for detailed deployment instructions, configuration options, and operational guidance.

- Claude apps gateway documentation — Complete setup guide including Docker Compose templates and IdP registration walkthroughs.
- Configuration reference — Full reference for all `gateway.yaml` options including telemetry destinations, upstream routing, and access policies.
- Deployment guide — Production deployment patterns, scaling, and operational best practices.

After the gateway is sending metrics to CloudWatch, view the dashboards as described in View the dashboards.

Set up OpenTelemetry for OpenAI Codex

OpenAI Codex emits OpenTelemetry (OTel) metrics about token usage, per-turn latency, API requests, tool calls, and approvals. When you send these metrics to Amazon CloudWatch, they populate the Codex tab of the Coding Agent Insights dashboards. For more information about the dashboards, see Coding Agent Insights.

Codex configures its metrics exporter through the `[otel]` section of its `config.toml` file. Two setup paths are available, depending on how your developers run Codex.

Choose a setup path

- **Bearer token** — Codex sends OTLP metrics directly to the CloudWatch native OTLP endpoint, authenticating with an `Authorization: Bearer` header. Identity and organizational attributes are supplied through the `OTEL_RESOURCE_ATTRIBUTES` environment variable. This path needs no collector and is the fastest way for an individual developer or a small team to get started. See Set up OpenAI Codex with a bearer token.
- **Enterprise rollout** — For organizations that federate developer identity through corporate single sign-on (SSO) such as Okta, Microsoft Entra ID, Auth0, or AWS IAM Identity Center, centralize

Amazon Bedrock access, and distribute a standard configuration to a fleet of developers. Identity and organizational attributes are derived from your identity provider (IdP). See Set up OpenAI Codex for an enterprise rollout.

> **Note**
>
> The bearer token path configures only how Codex sends *telemetry* to CloudWatch; it is independent of how Codex calls a model for *inference*, so you can use it with any model provider. The enterprise rollout, in contrast, is built around centralized Amazon Bedrock access, so Amazon Bedrock is the inference backend for that path. To use Amazon Bedrock for inference with Codex, see Amazon Bedrock in the OpenAI Codex documentation.

For the complete, continually updated guidance, including deployment templates and setup scripts, see the guidance for Codex on Amazon Bedrock on GitHub.

Topics

- Set up OpenAI Codex with a bearer token
- Set up OpenAI Codex for an enterprise rollout

Set up OpenAI Codex with a bearer token

On this path, OpenAI Codex sends OpenTelemetry (OTel) metrics directly to the Amazon CloudWatch native OTLP metrics endpoint and authenticates with a bearer token (a CloudWatch metrics API key). No OpenTelemetry collector is required. This is the fastest way for an individual developer or a small team to get Codex metrics into the Coding Agent Insights dashboards.

> **Important**
>
> Bearer tokens are long-term credentials. We recommend them only where short-term AWS credentials are not feasible, such as developer laptops outside AWS. For the security trade-offs and key rotation guidance, see Setting up bearer token authentication for Metrics. Organizations that federate identity through corporate SSO should use the enterprise rollout instead. See Set up OpenAI Codex for an enterprise rollout.

Prerequisites

- OpenAI Codex installed on the developer's machine.
- A working way for Codex to call a model for inference. This telemetry setup is independent of inference, so you can use either an OpenAI account (the default Codex provider) or Amazon Bedrock. To use Amazon Bedrock for inference, see Amazon Bedrock in the OpenAI Codex documentation.
- A CloudWatch metrics API key (bearer token), created from an IAM user that has CloudWatch metrics permissions. For the procedure that creates the IAM user and generates the key, see Setting up bearer token authentication for Metrics.

Configure the metrics exporter

Codex reads its configuration from `~/.codex/config.toml`. Add an `[otel.metrics_exporter]` section that points at the CloudWatch metrics OTLP endpoint and sends your bearer token in the `Authorization` header. Include the full `/v1/metrics` path; Codex does not append it for you.

```
[otel]
environment = "production"

[otel.metrics_exporter]
otlp-http = { endpoint = "https://monitoring.region.amazonaws.com/v1/metrics", protocol
 = "binary", headers = { "Authorization" = "Bearer YOUR_API_KEY" } }
```

> **Note**
>
> Some Codex versions do not expand environment variables inside header values, so the API key is shown inline. Treat `config.toml` as a secret: restrict its file permissions (for example, `chmod 600 ~/.codex/config.toml`) and do not commit it to source control.

Add identity and organizational attributes

The Coding Agent Insights dashboards group usage by developer, team, department, and other organizational dimensions. Provide these as OTel *resource* attributes through the `OTEL_RESOURCE_ATTRIBUTES` environment variable in the shell that launches Codex. Set the values from your developer's identity and your organization's structure.

```
export OTEL_RESOURCE_ATTRIBUTES="\
user.id=alias,\
user.email=developer@example.com,\
user.name=Jane Doe,\
department=engineering,\
team.id=platform,\
cost_center=cc-1234,\
organization=example-corp,\
location=us-seattle,\
role=software-engineer,\
manager=manager@example.com"
```

> **Note**
>
> Setting these as resource attributes (rather than per-datapoint attributes) matches the shape the dashboards expect, so you can group by `@resource.team.id`, `@resource.department`, and similar labels. Codex stamps metric-level dimensions such as `model` and `token_type` on the metrics automatically.

Verify and view

Run Codex for a few turns to generate metrics, and then confirm the data reached CloudWatch.

To verify metrics

1. Run a Codex session that performs some work (for example, ask it to run a command and edit a file).
2. In the CloudWatch console, choose **GenAI Observability**, **Coding Agent Insights**, and then the Codex tab. Metrics typically appear within a few minutes.
3. Optionally, query the metrics directly with PromQL. For more information, see PromQL querying.

For the complete, continually updated guidance, see the guidance for Codex on Amazon Bedrock on GitHub.

Set up OpenAI Codex for an enterprise rollout

Use this path when you operate Codex across a fleet of developers and want consistent, automatically attributed telemetry without distributing long-term credentials. Developers authenticate through your corporate identity provider (IdP), so identity and organizational attributes are derived from your directory rather than entered by hand. Metrics still arrive in Amazon CloudWatch with identity and organizational data carried as OTel *resource* attributes, so the same Coding Agent Insights dashboards populate.

This path supports corporate single sign-on (SSO) providers including Okta, Microsoft Entra ID (Azure AD), Auth0, and AWS IAM Identity Center.

How it works

In an enterprise rollout, a developer's identity flows from your IdP into the telemetry that reaches CloudWatch. A typical deployment includes the following components:

- **Identity federation** — Developers sign in through your IdP (Okta, Microsoft Entra ID, Auth0, or AWS IAM Identity Center). The IdP provides the developer's email, and organizational attributes such as department, team, and cost center.
- **Centralized Amazon Bedrock access** — Federated developers assume a role that grants access to Amazon Bedrock for inference, so you don't distribute static model credentials.
- **Local collector** — A per-developer OpenTelemetry collector receives metrics from Codex on `localhost`, stamps the identity and organizational resource attributes, signs requests with AWS Signature Version 4 (SigV4), and forwards them to the CloudWatch native OTLP endpoint. Because the collector signs with the developer's federated credentials, no bearer token is distributed.

Roll out to your fleet

The typical end-to-end rollout follows these stages.

To roll out Codex telemetry across your organization

1. **Configure identity federation.** Connect your IdP to AWS IAM Identity Center (or configure OIDC federation directly), and map the directory attributes you want to attribute usage by—such as department, team, and cost center—so they are available to the rollout.

2. **Centralize Amazon Bedrock access.** Grant federated developers a role that permits the Amazon Bedrock model invocations Codex needs, scoped to the AWS Regions and models you allow.
3. **Deploy the infrastructure.** Deploy the provided AWS CloudFormation templates to create the roles, policies, and supporting resources for the fleet.
4. **Generate and distribute developer configuration.** Use the provided setup scripts to generate each developer's collector configuration. The scripts derive `user.email` and `user.id` from the federated session and populate organizational resource attributes (department, team, cost center, organization, and similar) from your IdP. They then write the rendered collector configuration and the Codex `[otel]` block that points at the local collector.
5. **Developers run Codex.** Each developer signs in through SSO, starts the local collector, and runs Codex. Metrics flow from Codex to the local collector to CloudWatch, attributed to the developer and their organization automatically.

> **Note**
>
> On this path, identity and organizational attributes are emitted as OTel *resource* attributes, the same shape the bearer token path produces. As a result, the same Coding Agent Insights dashboards populate without additional configuration.

Complete guidance

For the complete, continually updated guidance—including the CloudFormation templates, identity-federation setup, and fleet setup scripts—see the guidance for Codex on Amazon Bedrock on GitHub. After the fleet is sending metrics, view the dashboards as described in View the dashboards.

Set up OpenTelemetry for GitHub Copilot

GitHub Copilot emits OpenTelemetry (OTel) metrics about token usage, requests, tool calls, and latency. When you send these metrics to Amazon CloudWatch, they populate the GitHub Copilot tab of the Coding Agent Insights dashboards. For more information about the dashboards, see Coding Agent Insights.

Copilot sends OTLP metrics directly to the CloudWatch native OTLP metrics endpoint and authenticates with a bearer token (a CloudWatch metrics API key). No OpenTelemetry collector is required. The bearer token is the supported path for Copilot today.

> **Note**
>
> This guide covers only how to emit Copilot's OTel metrics to CloudWatch, which is independent of how Copilot calls a model for inference. Copilot uses its own models by default, but enterprises can bring their own model provider keys—including Amazon Bedrock. For more information, see Use your own API keys in the GitHub Copilot documentation.

> **Important**
>
> Bearer tokens are long-term credentials. We recommend them only where short-term AWS credentials are not feasible, such as developer laptops outside AWS. For the security trade-offs and key rotation guidance, see Setting up bearer token authentication for Metrics.

Prerequisites

- GitHub Copilot installed, on a version that supports OpenTelemetry metrics export.
- An active GitHub Copilot subscription for model inference. Copilot uses its own models by default; to use Amazon Bedrock instead, configure a custom model key as described in Use your own API keys in the GitHub Copilot documentation.
- A CloudWatch metrics API key (bearer token), created from an IAM user that has CloudWatch metrics permissions. For the procedure that creates the IAM user and generates the key, see Setting up bearer token authentication for Metrics.

Configure the metrics exporter

Configure Copilot's OpenTelemetry exporter to send metrics to the CloudWatch metrics endpoint with your bearer token in the `Authorization` header. Set the following environment variables in the environment that launches Copilot. Set `COPILOT_OTEL_ENABLED=true` to turn on OpenTelemetry. Copilot uses the standard OpenTelemetry variables for all signals; it does not

support metrics-specific overrides. Set `OTEL_EXPORTER_OTLP_ENDPOINT` to the base endpoint *without* the `/v1/metrics` path — the exporter appends the signal path for you.

```
export COPILOT_OTEL_ENABLED=true
export OTEL_EXPORTER_OTLP_ENDPOINT="https://monitoring.region.amazonaws.com"
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer YOUR_API_KEY"
```

> **Note**
>
> Treat the bearer token as a secret. Avoid committing it to source control or to shared shell profiles, and rotate the key periodically as described in Setting up bearer token authentication for Metrics.

Add identity and organizational attributes

The Coding Agent Insights dashboards group usage by developer, team, department, and other organizational dimensions. Provide these as OTel *resource* attributes through the `OTEL_RESOURCE_ATTRIBUTES` environment variable. Set the values from your developer's identity and your organization's structure.

```
export OTEL_RESOURCE_ATTRIBUTES="\
user.id=alias,\
user.email=developer@example.com,\
user.name=Jane Doe,\
department=engineering,\
team.id=platform,\
cost_center=cc-1234,\
organization=example-corp,\
location=us-seattle,\
role=software-engineer,\
manager=manager@example.com"
```

Verify and view

Use Copilot to generate metrics, and then confirm the data reached CloudWatch.

To verify metrics

1. Use Copilot to perform some work that generates telemetry.

2. In the CloudWatch console, choose **GenAI Observability**, **Coding Agent Insights**, and then the GitHub Copilot tab. Metrics typically appear within a few minutes.
3. Optionally, query the metrics directly with PromQL. For more information, see PromQL querying.

CloudWatch investigations

The CloudWatch investigations feature is a generative AI-powered assistant that can help you respond to incidents in your system. It uses generative AI to scan your system's telemetry and quickly surface telemetry data and suggestions that might be related to your issue. These suggestions include metrics, logs, deployment events, and root-cause hypotheses with visual representations when multiple resources are involved. For a complete list of types of data that the AI assistant can surface, see Insights that CloudWatch investigations can surface in investigations.

You can conduct investigations without any additional configuration in CloudWatch operational troubleshooting. When you start an investigation, CloudWatch investigations uses the permissions associated with the signed-in user to investigate and analyze the resources associated with the alarm, metrics, or Logs Insights query and provide troubleshooting suggestions. No resources are created by the investigation, and every action taken by CloudWatch investigations is logged in CloudTrail for traceability. The investigation provides you the following information to assist you with operational troubleshooting:

- View AI-generated observations, suggestions, and hypotheses
- Access visual representations of multi-resource hypotheses
- Review natural language explanations and root cause analysis
- Access AI analysis of telemetry data, including metrics, logs, deployment events, AWS Health events, CloudTrail change events, X-Ray trace data, and CloudWatch Logs Insights queries

  Configuring CloudWatch investigations will provide you with more in depth investigations.

When you configure CloudWatch investigations, your investigations have the following additional capabilities:

- Accept or discard suggestions and observations

  For each suggestion, you decide whether to add it to the investigation findings or to discard it. This helps CloudWatch investigations refine and iterate toward the root cause of the issue. CloudWatch investigations can help you find the root cause without having to manually identify and query multiple metrics and other sources of telemetry and events. A troubleshooting issue that would have taken hours of searching and switching between different consoles can be solved in a much shorter time.

- Configure cross account access

  Use CloudWatch cross-account observability to enable investigation to collect data from other source accounts.

- Add new telemetry sources to the investigation

  Adding data from CloudTrail event history helps CloudWatch investigations associate issues with change events. Adding X-Ray provides improved topology and application mapping. You can also add data from Application Signals to dive deeper into the health of your applications and services, combining that telemetry with the other telemetry sources. If you use Amazon EKS Clusters you can provide CloudWatch investigations access to your EKS resources, to provide more granular information about the cluster resources that might be involved in the issue under investigation.

- Add notes or comments to investigation findings

  Being able to provide additional context to investigation finding to give perspective during reporting or auditing.

- Execute suggested runbook remediations

  CloudWatch investigations might suggest that you use an Automation runbook to attempt to automatically resolve the issue. Automation is a capability in Systems Manager, another AWS service. Automation runbooks define a series of steps, or actions, to be run on the resources that you select. Each runbook is designed to address a specific issue.

- Share investigation results with team members

  Without additional configuration investigation are linked with the signed-in user's session. Other user's can't view the investigation results or continue the investigation. After configuring CloudWatch investigations investigations are available to all users in the account that have been granted the required permissions.

- End, archive, or reopen the investigation manually

  Before CloudWatch investigations is configured in your account investigations run once and then complete. Once CloudWatch investigations is configured, investigations can continue until resolved. After the issue is resolved, the investigation is archived. If you have resolved the issue, but the conditions that caused the investigation are still present, you can manually close the investigation. If the conditions re-emerge you can restart (or reopen) the investigation.

- Investigation reporting

When you complete an investigation, you can generate a comprehensive investigation report that automatically captures all investigation findings, timeline events, and recommended actions.

Configuration of CloudWatch investigations creates an investigation group in your account. Each account can have one investigation group with up to 2 concurrent active investigations in the investigation group. Each month, each account can create up to 150 enhanced investigations with AI analysis. Investigation groups are account-level configurations. When an investigation group is created in an account, it is used with all investigations started in the account.

> **Note**
>
> When you configure CloudWatch investigations , CloudWatch will use the provided IAM role to periodically scan resources in your account for the purpose of mapping resources and telemetry. Some services like Lambda will invoke the KMS decrypt API on behalf of CloudWatch for certain API calls related to describing or listing resources. This background process is performed to make sure that the topology reflects the most recent state of the account and its dependencies. This refresh occurs regardless of whether there is an active investigation or not.

Topics

- Methods to create an investigation
- Understanding hypothesis visualizations
- How CloudWatch investigations finds data for suggestions
- Costs associated with CloudWatch investigations
- Insights that CloudWatch investigations can surface in investigations
- AWS services where investigations are supported
- Conduct an CloudWatch investigation without additional configuration
- Configure CloudWatch investigations
- (Recommended) Best practices to enhance investigations
- Investigate operational issues in your environment
- Cross-account investigations

- Generate incident reports
- Integrations with other systems
- Security in CloudWatch investigations
- CloudWatch investigations data retention
- Troubleshooting

Methods to create an investigation

You can create investigations in the following ways:

- From within many AWS consoles. For example, you can start an investigation when viewing a CloudWatch metric or alarm in the CloudWatch console, or from a Lambda function's **Monitor** tab on its properties page.
- By following a prompt in chat with CloudWatch investigations. You can start by asking questions like "Why is my Lambda function slow today?" or "What's wrong with my database?"
- By configuring a CloudWatch alarm action to automatically start an investigation when the alarm goes into ALARM state.

After you start an investigation with any of these methods, CloudWatch investigations scans your system to find telemetry that might be relevant to the situation, and also generates hypotheses based on what it finds. CloudWatch investigations surfaces both the telemetry data and the hypotheses. At any time after accepting a hypothesis, you can generate a comprehensive incident report that automatically captures the current investigation findings, timeline events, and recommended actions.

Understanding hypothesis visualizations

When CloudWatch investigations generates hypotheses that include multiple resources, the investigation view provides a visual representation of the causal relationships between those resources. This visual hypothesis view helps you quickly understand complex issues without reading lengthy text explanations.

The hypothesis visualization displays resources as nodes connected by the pathways identified by CloudWatch investigations. For example, if a hypothesis involves Lambda function A affecting DynamoDB table B, you'll see two nodes visualizing the relationship.

Key features of hypothesis visualizations:

- **Resource nodes** - Each AWS resource mentioned in the hypothesis appears as a distinct node, labeled with the resource type and identifier.
- **Connections** - Connections between nodes indicate the relationships that CloudWatch investigations has identified.
- **Visual context** - The layout helps you understand the scope and complexity of multi-resource issues at a glance.

This visual representation is particularly valuable for:

- Understanding distributed system failures that span multiple services
- Identifying upstream and downstream impact relationships
- Quickly assessing the scope of an issue before diving into detailed analysis

> **Note**
>
> Hypothesis visualizations are automatically generated when CloudWatch investigations identifies causal relationships between multiple resources.

How CloudWatch investigations finds data for suggestions

CloudWatch investigations uses a wide range of data sources to determine dependency relationships and plan analysis paths, including telemetry data configurations, service configurations, and observed relationships. These dependency relationships are found more easily if you use CloudWatch Application Signals and AWS X-Ray. When Application Signals and X-Ray aren't available, CloudWatch investigations will attempt to infer dependency relationships through co-occurring telemetry anomalies.

While CloudWatch investigations will continue to analyze telemetry data and provide suggestions without these features enabled, we strongly recommend that you enable the services and features listed in (Recommended) Best practices to enhance investigations for optimal quality and performance for CloudWatch investigations.

Important

To help CloudWatch investigations provide the most relevant information, certain content might be used from CloudWatch investigations, including but not limited to, questions that you ask CloudWatch investigations and its response, insights, user interactions, telemetry, and metadata for service improvements. Your trust and privacy, as well as the security of your content, is our highest priority. For more information, see **AWS Service Terms** and **AWS responsible AI policy**.

You can opt out of having your content collected to develop or improve the quality of CloudWatch investigations by creating an AI service opt-out policy for CloudWatch or AI Operations (aiops). For more information, see AI services opt-out policies in the AWS Organizations User Guide.

Costs associated with CloudWatch investigations

CloudWatch investigations might incur AWS service usage including telemetry and resource queries and other API usage. While the majority of these will not be charged to your AWS bill, there are exceptions including, but not limited to:

- CloudWatch APIs (`ListMetrics`, `GetDashboard`, `ListDashboards`, and `GetInsightRuleReport`)
- X-Ray APIs (`GetServiceGraph`, `GetTraceSummaries`, and `BatchGetTraces`)
- CloudWatch investigations also uses AWS Cloud Control APIs which might incur usage of AWS services such as Amazon Kinesis Data Streams and AWS Lambda.
- Additionally, if you choose to integrate Amazon Q Developer in chat applications you might incur usage of Amazon Simple Notification Service.

For usage of these services exceeding the AWS Free Tier, you will see charges on your AWS bill. These charges are expected to be minimal for normal usage of CloudWatch investigations. For more information, see Amazon Kinesis Data Streams pricing, AWS Lambda pricing for Automation, and Amazon Simple Notification Service pricing.

Insights that CloudWatch investigations can surface in investigations

CloudWatch investigations can surface the following types of items and add them to the **Suggestions** tab of an investigation. For hypotheses involving multiple resources, visual diagrams may also be provided to illustrate causal relationships.

- Hypotheses about root causes
- CloudWatch alarms, including both metric alarms and composite alarms
- CloudWatch metrics
- AWS Health events
- Change events logged in CloudTrail
- X-Ray trace data
- CloudWatch Logs Insights queries for log groups in the Standard log class
- CloudWatch Contributor Insights data
- CloudWatch Application Signals data
- CloudWatch Database Insights data

AWS services where investigations are supported

You can launch investigations from telemetry data (such as CloudWatch metrics, alarms, and logs), review generated anomaly signals, and explore hypotheses on investigations. CloudWatch investigations work best when helping you with automated troubleshooting guidance on the AWS services listed below:

- Amazon API Gateway
- AWS AppSync
- Amazon Data Firehose
- Amazon DynamoDB
- Amazon EBS[1]
- Amazon EC2[1]
- Amazon ECS on Amazon EC2[2]

- Amazon ECS on Fargate[2]
- Amazon EKS[3]
- Amazon Kinesis Data Streams
- AWS Lambda
- Amazon OpenSearch Service
- Amazon RDS[4]
- Amazon S3
- Amazon SNS
- Amazon SQS
- AWS Step Functions

The list of services will continue to be expanded over time. CloudWatch investigations uses a wide range of data sources to determine dependency relationships and plan analysis paths, including telemetry data configurations, service configurations, and observed relationships through CloudWatch Application Signals and X-Ray. Where none of the above is available, CloudWatch investigations will attempt to infer dependency relationships through co-occurring telemetry anomalies.

Best practice setup

While CloudWatch investigations will continue to analyze telemetry data and provide suggestions without the following features enabled, for optimal quality and performance for CloudWatch investigations, we recommend you complete the following steps:

- [1]For both Amazon EC2 and Amazon EBS, update your CloudWatch agent to version 1.30049.1 or later. For more information, see Collect metrics, logs, and traces using the CloudWatch agent.
- [2]For Amazon ECS, enable CloudWatch Container Insights. For more information, see Container Insights.
- [3]For Amazon EKS, enable CloudWatch Container Insights and configure Amazon EKS Access Entries. For more information, see Container Insights and Integration with Amazon EKS.
- [4]For Amazon RDS, enable CloudWatch Database Insights in the **Advanced mode**. For more information, see Turning on the Advanced mode of Database Insights for Amazon Aurora in the *Aurora User Guide*.
- Enable CloudWatch Application Signals and X-Ray. For more information, see Application Signals and What is AWS X-Ray.

Conduct an CloudWatch investigation without additional configuration

You can conduct a CloudWatch investigations AI-powered root cause analysis without any additional configuration of your AWS account using the **Investigations** feature available in **Operational troubleshooting**.

These investigations provide instant access to CloudWatch investigations capabilities from commonly used telemetry sources in the AWS console. This type of investigation is session-based, read-only, and automatically deleted after 24 hours.

The investigation pane provides you:

- **AI-generated observations** - Telemetry that fans out from your initial observation you provided to locate additional supporting data that can lead to a potential root cause.
- **Root cause hypotheses** - Potential explanations for what's causing the problem, including causal diagrams when multiple resources are involved
- **Natural language explanations** - Easy-to-understand descriptions of findings and recommendations

You can use the suggestions from the investigation pane to help accelerate your troubleshooting process.

To start an investigation

1. Navigate to any supported telemetry source showing an issue you want to investigate, such as:
   - A CloudWatch metric showing unusual patterns
   - A CloudWatch alarm in ALARM state
   - A Lambda function's monitoring tab showing performance issues
2. In **Operational troubleshooting**, choose the **Investigate** tab, then choose **Start investigation**.
3. Watch as CloudWatch investigations analyzes your telemetry and generates insights about potential root causes.
4. (Optional) To add additional telemetry sources or collaborate on the investigation with others, choose **Get started** in the information box at the bottom of the **Investigate** tab. You'll be

guided through the investigation group configuration process. For more information, see Configure CloudWatch investigations

> **Important**
>
> You must have the appropriate IAM permissions to create or access investigation groups. Users with read-only permissions will see information about requesting additional permissions from their administrators.

Configure CloudWatch investigations

Configuring CloudWatch investigations creates an investigation group that CloudWatch investigations will use to access telemetry and aggregate data related to the investigation. Before you create the investigation group, you can walk-through a sample investigation to get an overall idea of how they work.

Topics

- See a sample investigation
- Set up an investigation group
- Configure alarms to create investigations
- Integration with Amazon EKS

See a sample investigation

If you'd like to see an investigation in action before you configure an investigation group for your account, you can walk through a sample investigation. The sample investigation doesn't use your data and doesn't make data calls or start API operations in your account.

To view the sample investigation

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Overview**.
3. Choose **Try a sample investigation**.

   The console displays the sample investigation, with suggestions and findings in the right pane. In each popup, choose **Next** to advance to the next part of the sample walkthrough.

Set up an investigation group

To set up CloudWatch investigations in your account for use with an enhanced investigation, you create an *investigation group*. Creating an investigation group is a one-time setup task, after it's created it is used to conduct other investigations. Settings in the investigation group help you centrally manage the common properties of your investigations, such as the following:

- Who can access the investigations
- Cross-account investigation support to access resources in other accounts during the investigation
- Whether investigation data is encrypted with a customer managed AWS Key Management Service key.
- How long investigations and their data are retained by default.

You can have one investigation group per account. Each investigation in your account will be part of this investigation group.

To create an investigation group you must be signed in to an IAM principal that has the either the **AIOpsConsoleAdminPolicy** or the **AdministratorAccess** IAM policy attached, or to an account that has similar permissions.

> **Note**
>
> To be able to choose the recommended option of creating a new IAM role for CloudWatch investigations operational investigations, you must be signed in to an IAM principal that has the `iam:CreateRole`, `iam:AttachRolePolicy`, and `iam:PutRolePolicy` permissions.

> **Important**
>
> CloudWatch investigations uses *cross-Region inference* to distribute traffic across different AWS Regions. For more information, see Cross-Region inference.

To create an investigation group in your account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the left navigation pane, choose **AI Operations**, **Configuration**.
3. Choose **Configure for this account**.
4. Optionally change the retention period for investigations. For more information about what the retention period governs, see CloudWatch investigations data retention.
5. (Optional) To encrypt your investigation data with a customer managed AWS KMS key, choose **Customize encryption settings** and follow the steps to create or specify a key to use. If you don't specify a customer managed key, CloudWatch investigations uses an AWS owned key for encryption. For more information, see Encryption of investigation data.
6. Choose how to give CloudWatch investigations permissions to access resources. To be able to choose either of the first two options, you must be signed in to an IAM principal that has the `iam:CreateRole`, `iam:AttachRolePolicy`, and `iam:PutRolePolicy` permissions.

   a. (Recommended) Select **Auto-create a new role with default investigation permissions**. This role will be granted permissions using the AWS managed policies for AI Operations.For more information, see User permissions for your CloudWatch investigations group.

   b. Create a new role yourself and then assign the policy templates.

   c. Choose **Assign an existing role** if you already have a role with the permissions that you want to use.

      If you choose this option, you must make sure the role includes a trust policy that names `aiops.amazonaws.com` as the service principal. For more information about using service principals in trust policies, see AWS service principals.

      We also recommend that you include a `Condition` section with the account number, to prevent a confused deputy situation. The following example trust policy illustrates both the service principal and the `Condition` section.

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "aiops.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:aiops:us-
east-1:123456789012:*"
                }
            }
        }
    ]
}
```

7. Choose **Create investigation group**, you can now create an investigation from an alarm, metric, or log insight.

Optionally, you can setup additional recommended configurations to enhance your experience.

1. In the left navigation pane, choose **AI Operations, Configuration**.
2. On the **Optional configuration** tab, choose the enhancements you want to add to CloudWatch investigations.
3. In **Configure cross account access** you can set this account as a monitoring account that collects data from other source accounts in your organization. For more information, see Cross-account investigations.
4. For **Enhanced integrations**, choose to allow CloudWatch investigations access to additional services in your system, to enable it to gather more data and be more useful.

    a. In the **Tags for application boundary detection** section, enter the existing custom tag keys for custom applications in your system. Resource tags help CloudWatch investigations narrow the search space when it is unable to discover definite relationships between resources. For example, to discover that an Amazon ECS service depends on an Amazon RDS database, CloudWatch investigations can discover this relationship using data sources such as X-Ray and CloudWatch Application Signals. However, if you haven't deployed these features, CloudWatch investigations will attempt to identify possible relationships. Tag boundaries can be used to narrow the resources that will be discovered by CloudWatch investigations in these cases.

You don't need to enter tags created by myApplications or CloudFormation, because CloudWatch investigations can automatically detect those tags.

b. CloudTrail records events about changes in your system including deployment events. These events can often be useful to CloudWatch investigations to create hypotheses about root causes of issues in your system. In the **CloudTrail for change event detection** section, you can give CloudWatch investigations some access to the events logged by AWS CloudTrail by enabling **Allow the assistant access to CloudTrail change events through the CloudTrail Event history**. For more information, see Working with CloudTrail Event history.

c. The **X-Ray for topology mapping** and **Application Signals for health assessment** sections point out other AWS services that can help CloudWatch investigations find information. If you have deployed them and you have granted the **AIOpsAssistantPolicy** IAM policy to CloudWatch investigations, it will be able to access X-Ray and Application Signals telemetry.

   For more information about how these services help CloudWatch investigations, see X-Ray and CloudWatch Application Signals.

d. If you use Amazon EKS, your CloudWatch investigations investigation group can utilize information directly from your Amazon EKS cluster after you set up access entries. For more information, see Integration with Amazon EKS.

e. If you use Amazon RDS, enable the Advanced mode of Database Insights on your database instances. Database Insights monitors database load and provides detailed performance analysis that helps CloudWatch investigations identify database-related issues during investigations. When Advanced Database Insights is enabled, CloudWatch investigations can automatically generate performance analysis reports that include detailed observations, metric anomalies, root cause analysis, and recommendations specific to your database workload. For more information about Database Insights and how to enable Advanced mode, see Monitoring Amazon RDS databases with CloudWatch Database Insights.

5. You can integrate CloudWatch investigations with a *chat channel*. This makes it possible to receive notifications about an investigation through the chat channel. CloudWatch investigations support chat channels in the following applications:

   - Slack
   - Microsoft Teams

If you want to integrate with a chat channel, we recommend that you complete some additional steps before enabling this enhancement in your investigation group. For more information, see Integration with third-party chat systems.

Then, perform the following steps to integrate with a chat channel in chat applications:

- In the **Chat client integration** section, choose **Select SNS topic**.
- Select the SNS topic to use for sending notifications about your investigations.

Configure alarms to create investigations

You can configure an existing CloudWatch alarm to automatically create investigations in CloudWatch investigations. When the alarm enters the ALARM state, CloudWatch automatically creates a new investigation or adds to an existing investigation based on the deduplication string.

When configuring an alarm to automatically create investigations, you'll need to specify an Amazon Resource Name (ARN) in the alarm's actionArns. This ARN identifies the investigation group where alarm-triggered investigations will be created. You can optionally include a deduplication string in the ARN to group related alarms.

ARN format and parameters

The ARN pattern for investigation group alarm actions follows this format:

```
arn:aws:aiops:region:account-id:investigation-group/investigation-group-
identifier#DEDUPE_STRING=value
```

The following table describes each ARN component:

Parameter	Description
`region` (required)	The AWS Region where your investigation group is located. For example: `us-east-1` .
`account-id` (required)	Your 12-digit AWS account ID. For example: `123456789012` .

Parameter	Description
`investigation-group-identifier` (required)	The unique identifier of your investigation group. Fore example, `sMwwg1IogXdvL7UZ`
`DEDUPE_STRING=value` (optional)	A deduplication string that groups related alarms into the same investigation. When multiple alarms use the same deduplication string, they contribute to a single investigation instead of creating separate ones.

Example without deduplication string:

```
arn:aws:aiops:us-east-1:123456789012:investigation-group/sMwwg1IogXdvL7UZ
```

Example with deduplication string:

```
arn:aws:aiops:us-east-1:123456789012:investigation-group/
sMwwg1IogXdvL7UZ#DEDUPE_STRING=performance
```

Benefits of deduplication strings

Deduplication strings help you organize related alarms and reduce investigation fragmentation. Use deduplication strings when:

- **Multiple alarms monitor the same system** - CPU, memory, and disk alarms for the same EC2 instance can share a deduplication string to create one comprehensive investigation.
- **Cascading failures occur** - When one issue triggers multiple related alarms, the same deduplication string prevents creating separate investigations for each symptom.
- **You want to categorize by problem type** - Use descriptive strings like "performance", "connectivity", or "security" to group alarms by issue category.

Effective deduplication string examples:

- `DEDUPE_STRING=webserver-performance` - Groups performance-related alarms for web servers
- `DEDUPE_STRING=database-connectivity` - Groups database connection issues

- `DEDUPE_STRING=instance-i-1234567890abcdef0` - Groups all alarms for a specific EC2 instance

> **Note**
>
> If no deduplication string is specified, the system uses a default combination of alarm name, account ID, and region to group investigations.

For more information about investigation groups, see Set up an investigation group.

Configure an alarm to create investigations

After you have an investigation group set up in your account, you can configure existing CloudWatch alarms to automatically create investigations when they enter the ALARM state. This eliminates the need to manually start investigations and ensures consistent response to operational issues. You can configure alarms using the AWS Management Console, AWS CLI, CloudFormation, or AWS SDKs.

Console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, and select an existing alarm.
3. Choose **Actions**, **Edit**.
4. In the **Alarm actions** section, choose **Add alarm action**.
5. Under the **Configure actions**, **Investigation action** section, choose the investigation group ARN.
6. (Optional) Add a deduplication string to group related alarms.
7. Choose **Update alarm**.

CLI

This command requires that you specify an ARN for the `alarm-actions` parameter. For information about how to create the ARN, see ARN format and parameters.

To configure a CloudWatch alarm with InvestigationGroup action (AWS CLI)

1. Install and configure the AWS CLI, if you haven't already. For information, see Installing or updating the latest version of the AWS CLI.
2. Run the following command to collect information about the alarm that you want to configure.

```
aws cloudwatch describe-alarms --alarm-names "alarm name"
```

3. Run the following command to update an alarm. Replace each *example resource placeholder* with your own information.

```
aws cloudwatch put-metric-alarm --alarm-name name \
--alarm-description "description" \
--metric-name name --namespace namespace \
--statistic statistic --period value --threshold value \
--comparison-operator value \
--dimensions "dimensions" --evaluation-periods value \
--alarm-actions "arn:aws:aiops:region:{account-id}:investigation-group/
{investigationGroupIdentifier}#DEDUPE_STRING={my-dedupe-string}"
```

Here's an example.

```
//Without deduplication string
aws cloudwatch put-metric-alarm --alarm-name cpu-mon \
--alarm-description "Alarm when CPU exceeds 70 percent" \
--metric-name CPUUtilization --namespace AWS/EC2 \
--statistic Average --period 300 --threshold 70 \
--comparison-operator GreaterThanThreshold \
--dimensions "Name=InstanceId,Value=i-12345678" --evaluation-periods 2 \
--alarm-actions arn:aws:aiops:us-east-1:123456789012:investigation-group/
sMwwg1IogXdvL7UZ \
--unit Percent

//With deduplication string
aws cloudwatch put-metric-alarm --alarm-name cpu-mon \
--alarm-description "Alarm when CPU exceeds 70 percent" \
--metric-name CPUUtilization --namespace AWS/EC2 \
--statistic Average --period 300 --threshold 70 \
--comparison-operator GreaterThanThreshold \
--dimensions "Name=InstanceId,Value=i-12345678" --evaluation-periods 2 \
```

```
--alarm-actions arn:aws:aiops:us-east-1:123456789012:investigation-group/
sMwwg1IogXdvL7UZ#DEDUPE_STRING=performance \
--unit Percent
```

CloudFormation

This section includes CloudFormation templates that you can use to configure CloudWatch alarms to automatically create or update investigations. Each template requires that you specify an ARN for the `AlarmActions` parameter. For information about how to create the ARN, see ARN format and parameters.

```
//Without deduplication string
Resources:
  MyAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmActions:
        - !Sub "arn:aws:aiops:${AWS::Region}:${AWS::AccountId}:investigation-group/
{investigationGroupIdentifier}"

//With deduplication string
Resources:
  MyAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmActions:
        - !Sub "arn:aws:aiops:${AWS::Region}:${AWS::AccountId}:investigation-group/
{investigationGroupIdentifier}#DEDUPE_STRING={my-dedupe-string}"
```

SDK

This section includes Java code snippets that you can use to configure CloudWatch alarms to automatically create or update investigations. Each snippet requires that you specify an ARN for the `investigationGroupArn` parameter. For information about how to create the ARN, see ARN format and parameters.

```
import com.amazonaws.services.cloudwatch.AmazonCloudWatch;
import com.amazonaws.services.cloudwatch.AmazonCloudWatchClientBuilder;
import com.amazonaws.services.cloudwatch.model.ComparisonOperator;
import com.amazonaws.services.cloudwatch.model.Dimension;
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmRequest;
```

```
import com.amazonaws.services.cloudwatch.model.PutMetricAlarmResult;
import com.amazonaws.services.cloudwatch.model.StandardUnit;
import com.amazonaws.services.cloudwatch.model.Statistic;

//Without deduplication string
private void putMetricAlarmWithCloudWatchInvestigationAction() {
        final AmazonCloudWatch cloudWatchClient =
                AmazonCloudWatchClientBuilder.defaultClient();

        Dimension dimension = new Dimension()
                .withName("InstanceId")
                .withValue("i-12345678");
        String investigationGroupArn = "arn:aws:aiops:us-
east-1:123456789012:investigation-group/sMwwg1IogXdvL7UZ";

        PutMetricAlarmRequest request = new PutMetricAlarmRequest()
                    .withAlarmName("cpu-mon")
                    .withComparisonOperator(
                        ComparisonOperator.GreaterThanThreshold)
                    .withEvaluationPeriods(2)
                    .withMetricName("CPUUtilization")
                    .withNamespace("AWS/EC2")
                    .withPeriod(300)
                    .withStatistic(Statistic.Average)
                    .withThreshold(70.0)
                    .withActionsEnabled(true)
                    .withAlarmDescription("Alarm when CPU exceeds 70 percent")
                    .withUnit(StandardUnit.Percent)
                    .withDimensions(dimension)
                    .withAlarmActions(investigationGroupArn);

        PutMetricAlarmResult response = cloudWatchClient.putMetricAlarm(request);
}

//With deduplication string
private void putMetricAlarmWithCloudWatchInvestigationActionWithDedupeString() {
        final AmazonCloudWatch cloudWatchClient =
                AmazonCloudWatchClientBuilder.defaultClient();

        Dimension dimension = new Dimension()
                .withName("InstanceId")
                .withValue("i-12345678");
        String investigationGroupArn = "arn:aws:aiops:us-
east-1:123456789012:investigation-group/sMwwg1IogXdvL7UZ#DEDUPE_STRING=performance";
```

```
        PutMetricAlarmRequest request = new PutMetricAlarmRequest()
                    .withAlarmName("cpu-mon")
                    .withComparisonOperator(
                        ComparisonOperator.GreaterThanThreshold)
                    .withEvaluationPeriods(2)
                    .withMetricName("CPUUtilization")
                    .withNamespace("AWS/EC2")
                    .withPeriod(300)
                    .withStatistic(Statistic.Average)
                    .withThreshold(70.0)
                    .withActionsEnabled(true)
                    .withAlarmDescription("Alarm when CPU exceeds 70 percent")
                    .withUnit(StandardUnit.Percent)
                    .withDimensions(dimension)
                    .withAlarmActions(investigationGroupArn);

        PutMetricAlarmResult response = cloudWatchClient.putMetricAlarm(request);
}
```

Integration with Amazon EKS

CloudWatch investigations investigation groups can utilize information directly from your Amazon EKS cluster. To get started, first grant access to the `Investigation Group` IAM role. We recommend using the default AWS managed *access policy* `AmazonAIOpsAssistantPolicy` that grants CloudWatch investigations investigation groups access to resources in the cluster. By using this policy you will automatically get policy updates as needed.

> **Note**
>
> `AmazonAIOpsAssistantPolicy` is an access policy. The AWS managed identity policy that authorizes the access associated with CloudWatch investigations investigation groups is `AIOpsAssistantPolicy`.

Use the **Advanced Configuration** option to scope down the access provided by the access policy to a set of namespaces or the entire cluster. Alternatively, you can further scope access down by associating the access entry to a Kubernetes group RBAC permission. For more information, see Creating access entries.

Configuring the Amazon EKS access entry (Console)

To associate the `AmazonAIOpsAssistantPolicy` to the investigation role using the AWS Management Console, follow these steps:

1. Open the CloudWatch console and navigate to the Investigations Configuration page.
2. In the Amazon EKS Access section, select the option to associate the `AmazonAIOpsAssistantPolicy` with your investigation role.
3. Review the policy details and confirm the association.

To further customize the access scope:

1. Click **Advanced Configuration** in the Amazon EKS Access section.
2. You will be redirected to the Amazon EKS console.
3. In the Amazon EKS console, you can:
    a. Scope the policy to specific namespaces
    b. Configure the group feature for more granular access control

Configuring Amazon EKS Access Entries (CDK)

To configure Amazon EKS Access Entries using the AWS CDK, use the following code example:

```
const testAccessEntry = new AccessEntry(this, `test-access-entry`, {
    cluster: eksCluster,
    principal: investigationsIamRole.roleArn,
    accessPolicies: [
        AccessPolicy.fromAccessPolicyName('AmazonAIOpsAssistantPolicy', {
            accessScopeType: AccessScopeType.CLUSTER
        }),
    ],
});
```

AmazonAIOpsAssistantPolicy

The Amazon EKS Access Policy, `AmazonAIOpsAssistantPolicy`, provides comprehensive Read Only access to resources in the cluster. Information from each resource may not be currently utilized by CloudWatch investigations.

```
- apiGroups: [""]
  resources:
    - pods
    - pods/log
    - services
    - nodes
    - namespaces
    - events
    - persistentvolumes
    - persistentvolumeclaims
    - configmaps
  verbs:
    - get
    - list

- apiGroups: ["apps"]
  resources:
    - deployments
    - replicasets
    - statefulsets
    - daemonsets
  verbs:
    - get
    - list

- apiGroups: ["batch"]
  resources:
    - jobs
    - cronjobs
  verbs:
    - get
    - list

- apiGroups: ["events.k8s.io"]
  resources:
    - events
```

```
      verbs:
        - get
        - list

    - apiGroups: ["networking.k8s.io"]
      resources:
        - ingresses
        - ingressclasses
      verbs:
        - get
        - list

    - apiGroups: ["storage.k8s.io"]
      resources:
        - storageclasses
      verbs:
        - get
        - list

    - apiGroups: ["metrics.k8s.io"]
      resources:
        - pods
        - nodes
      verbs:
        - get
        - list
```

Updates to AmazonAIOpsAssistantPolicy

Change	Description	Date
Add policy for CloudWatch investigations	Initial release of `AmazonAIOpsAssistantPolicy`	August 9, 2025

(Recommended) Best practices to enhance investigations

As a best practice, we recommend that you enable several AWS services and features in your account that can help CloudWatch investigations discover more information in your topology and make better suggestions during investigations.

Topics

- CloudWatch agent
- AWS CloudTrail
- CloudWatch Application Signals
- X-Ray
- Container insights
- Database insights

CloudWatch agent

We recommend that you install the latest version of the CloudWatch agent on your servers. Using a recent version of the CloudWatch agent enhances the ability to find issues in Amazon EC2 and Amazon EBS during investigations. At a minimum, you should use Version 1.300049.1 or later of the CloudWatch agent. For more information about the CloudWatch agent and how to install it, see Collect metrics, logs, and traces using the CloudWatch agent.

AWS CloudTrail

We recommend that you configure your CloudTrail trails to send management events to CloudWatch Logs. CloudTrail records management events about control plane operations in your AWS account, such as configuring permissions policies and creating, modifying, and updating resources. When CloudTrail events are sent to CloudWatch Logs, CloudWatch investigations can analyze these events in your AWS account to detect changes in your account related to your investigation. For more information, see What is AWS CloudTrail, Creating a trail with the CloudTrail console, Working with CloudTrail trails and Sending events to CloudWatch Logs.

> **Note**
>
> CloudWatch investigations can only analyze CloudTrail events from trails that send events to CloudWatch Logs in the same AWS Region where you conduct the investigation. For cross-account investigations, it will review the CloudTrail trails in each configured account in the same region where you conduct the investigation.

CloudWatch Application Signals

CloudWatch Application Signals discovers the topology of your environment, including your applications and their dependencies. It also automatically collects standard metrics such as latency and availability. By enabling Application Signals, CloudWatch investigations can use this topology and metric information during investigations.

For more information about application signals, see Application Signals.

X-Ray

We recommend that you enable AWS X-Ray. X-Ray collects traces about requests that your applications serve. For any traced request to your application, you can see detailed information not only about the request and response, but also about calls that your application makes to downstream AWS resources, microservices, databases, and web APIs. This information can help CloudWatch investigations during investigations.

For more information, see **What is AWS X-Ray**

Container insights

If you use Amazon ECS or Amazon EKS, we recommend that you install Container insights. This improves the ability of CloudWatch investigations to find issues in your containers. For more information about the CloudWatch agent and how to install it, see Container Insights.

Database insights

If you use Amazon RDS, we recommend that you enable the Advanced mode of Database Insights on your database instances. Database Insights monitors database load and provides detailed performance analysis that helps CloudWatch investigations identify database-related issues during investigations. When Advanced Database Insights is enabled, CloudWatch investigations can automatically generate performance analysis reports that include detailed observations, metric anomalies, root cause analysis, and recommendations specific to your database workload. For more information about Database Insights and how to enable Advanced mode, see Monitoring Amazon RDS databases with CloudWatch Database Insights.

Investigate operational issues in your environment

You can create investigations in several ways depending on your workflow and the source of the issue you're investigating. After an investigation is active, you can review AI-generated suggestions, accept or discard findings, and take remediation actions through automated runbooks.

The following procedures show you how to start investigations from different entry points and how to work with active investigations:

Contents

- Create an investigation
  - Create an investigation from Amazon Q chat
  - Create an investigation from a CloudWatch alarm action
- Create an investigation from a CloudWatch Application Signals Service Level Objective (SLO)
- View and continue an open investigation
- Reviewing and executing suggested runbook remediations for CloudWatch investigations
- Manage your current investigations
- Restart an archived investigation

Create an investigation

You can start an investigation from several AWS consoles, including (but not limited to) CloudWatch alarm pages, CloudWatch metric pages, and Lambda monitoring pages.

To start an investigation from an AWS console page

1. In **Account-level** select the graph of the metric or alarm that you want to investigate.
2. If the top of the page has an **Investigate** button, choose it and then choose **Start new investigation**.

   Otherwise, choose the vertical ellipsis menu icon

   for the metric, and choose **Investigate**, **Start a new investigation**.
3. In the **Investigation** pane, enter a name for the investigation in **New investigation title**, and optionally enter notes about the selected metric or alarm.

4. Under **Approximate impact start time** CloudWatch investigations recommends a timestamp to investigate based on the selected telemetry. To change the timestamp of the investigation, update the date and time.
5. Then choose **Start investigation**.

   The investigation starts. CloudWatch investigations scans your telemetry data to find data that might be associated with this situation.
6. To move the investigation data to the larger pane, choose **Open in full page**.
7. For detailed instructions about steps that you can take while continuing the investigation, see View and continue an open investigation.

Create an investigation from Amazon Q chat

You can ask questions about issues in your deployment in CloudWatch investigations chat. The question could be something like "Why is my Lambda function slow today?"

When you do so, CloudWatch investigations might ask follow up questions and run a health check regarding the issue. After the health check, the chat will prompt you about whether you want to start an investigation.

For more information and more sample questions, see Chatting with Amazon Q about AWS.

For detailed instructions about steps that you can take while continuing the investigation after it has been started, see View and continue an open investigation.

Create an investigation from a CloudWatch alarm action

When you create a CloudWatch alarm, you can specify for it to automatically start an investigation when it goes into ALARM state. You can do this for both metric alarms and composite alarms. For more information, see Start a CloudWatch investigations from an alarm, Create a CloudWatch alarm based on a static threshold and Create a composite alarm.

Create an investigation from a CloudWatch Application Signals Service Level Objective (SLO)

You can start an investigation from a CloudWatch Application Signals Service Level Objective (SLO) metric.

To start an investigation from a CloudWatch Application Signals Service Level Objective (SLO)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Navigate to the **Applications Signals (APM)**, **Service Level Objectives (SLO)** console page.
3. Select an entry from the **Service Level Objectives (SLO)** list to display the metrics available for that SLO.
4. Select a metric, then choose **Investigate** from the **Action** menu.

   Alternatively, in the visualization of the metric you want to investigate, next to the more menu, select the AI icon to start an investigation.

   > **Note**
   >
   > If you have not configured operational investigations in your account, the AI icon opens the **Operation troubleshooting** pane. Select **Get started** to configure an investigation group and then continue.

5. In the **Operational troubleshooting** pane on the **Investigate**, under **Investigation title** enter a name for the investigation and optionally enter notes about the selected metric.
6. Under **Approximate impact start time** CloudWatch investigations recommends a timestamp to investigate based on the selected telemetry. To change the timestamp of the investigation, update the date and time.
7. Then choose **Start investigation**.

   The investigation starts. CloudWatch investigations scans your telemetry data to find data that might be associated with this situation.
8. To move the investigation data to the larger pane, choose **Open in full page**.
9. For detailed instructions about steps that you can take while continuing the investigation, see View and continue an open investigation.

View and continue an open investigation

Use the steps in this section to view and continue and existing investigation

To view and continue an investigation

1. If you aren't already on the page for the investigation, do the following:

   a. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   b. In the left navigation pane, choose **AI Operations**, **Investigations**.

   c. Choose the name of the investigation.

2. The **Feed** section displays the items that have been added to the investigation findings, including the metric or alarm that was originally selected to start the investigation with.

   The pane on the right includes tabs. Choose the **Suggestions** tab.

3. The **Suggestions** tab displays *observations* of other telemetry that CloudWatch investigations has found that might be related to the investigation. It might also include *hypotheses*, which are possible reasons or root causes that CloudWatch investigations has found for the situation.

   Both observations and hypotheses are written in natural language by CloudWatch investigations.

   You have several options:

   - For each suggestion, you can choose **Accept** or **Discard**.

     When you choose **Accept**, the suggestion is added to the **Feed** section, and CloudWatch investigations uses this information to direct further scanning and suggestions.

     If you choose **Discard**, the suggestion is moved to the **Discarded** tab.

   - For each observation-type suggestion, you can choose to expand the graph in the **Suggestions** tab, or open it in the CloudWatch console to see more details about it.

   - Some of the observations might be results of CloudWatch Logs Insights queries that CloudWatch investigations ran as part of the investigation. When an observation is a CloudWatch Logs Insights query result, the query itself is displayed as part of the observation. You can edit the query and re-run it. To do so, choose the vertical ellipsis menu icon 

     by the results, and then choose **Open in Logs Insights**. For more information, see Analyzing log data with CloudWatch Logs Insights.

- If you know of telemetry in an AWS service that might apply to this investigation, you can go to that service's console and add the telemetry to the investigation. For example, to add a Lambda metric to the investigation, you can do the following:

  1. Open the Lambda console.
  2. In the **Monitor** section, find the metric.
  3. Open the vertical ellipsis context menu
     
     for the metric, choose **Investigate**, **Add to investigation** Then, in the **Investigate** pane, select the name of the investigation.

- When you view a hypothesis in the **Suggestions** tab, you can choose **Show reasoning** to display the data that CloudWatch investigations used to generate the hypothesis. For hypotheses involving multiple resources, you may also see a visual representation showing the causal relationships between the resources as connected nodes.
- You can choose the **Discarded** tab and view the suggestions that have been previously discarded. To add one of them to the findings, choose **Restore to findings**.
- To add notes to the findings, choose **New note** in the **Feed** pane. Then enter your notes and choose **Add**.

4. When you add a hypothesis to the **Feed** area, it might display **Show suggested actions**. If so, choosing this displays possible actions that you can take, assuming that hypothesis is correct about the issue. Possible actions include the following:

   - **Documentation suggestions** are links to AWS documentation that can help you understand the issue that you are working on, and how to solve it. To view suggested documentation, choose its **Review** link
   - **Runbook suggestions** are suggestions that use the pre-defined *runbooks* in Systems Manager Automation. Each runbook defines a number of steps for performing a task on an AWS resource. For information about continuing with a runbook action, see Reviewing and executing suggested runbook remediations for CloudWatch investigations.

     > **Important**
     >
     > There is a charge for executing an Automation runbook. However, CloudWatch investigations provides you with a preview of actions that a suggested runbook takes, giving you an opportunity to better evaluate whether to execute the runbook.

For information about Automation pricing, see AWS Systems Manager pricing for Automation.

5. (Optional) Choose **Incident report** to create a comprehensive incident analysis document. For more information, see Generate incident reports.
6. When you are ready to end an investigation, choose **End investigation** and then optionally add final notes. The investigation status changes to **Archived**. You can restart archived investigations by opening the investigation page and choosing **Restart investigation**.

> **Note**
>
> At some point, you might see **Completed the analysis. Finished with the investigation.** displayed above the **Feed** area. If you then add more telemetry to the findings, this message changes and CloudWatch investigations begins scanning your telemetry again, based on the new data that you added to the findings.

Reviewing and executing suggested runbook remediations for CloudWatch investigations

When you add a hypothesis to the **Feed** area of an active investigation, CloudWatch investigations might display **Show suggested actions**. One suggested action might be to view documentation with information to help you remediate a problem manually.

Another suggestion might be to use an *Automation runbook* to attempt to automatically resolve the issue. Automation is a capability in Systems Manager, another AWS service. Automation runbooks define a series of steps, or actions, to be run on the resources that you select. Each runbook is designed to address a specific issue. Runbooks can address a variety of operational needs: Creating, repairing, reconfiguring, installing, troubleshooting, remediating, duplicating, and more. For more information about Automation, see Integration with AWS Systems Manager Automation.

Before you begin

Before working with Automation runbooks in an investigation, be aware of the following important considerations:

- Choosing to execute a runbook incurs charges. For information, see AWS Systems Manager pricing.
- Root causes and runbook suggestions are powered by automated reasoning and generative artificial intelligence services.

> **Important**
>
> You are responsible for actions that result from executing runbook steps and the choice of parameter values entered during runbook execution. You might need to edit the suggested runbook to make sure the runbook performs as expected. For more information, see **AWS responsible AI policy**.

- Depending on the runbook, you might need to enter values for the runbook's **Input parameters** before the execution can run.
- The runbook executes using the IAM permissions assigned to the operator. If necessary, sign in with different IAM permissions to execute the runbook. In addition to permissions for the actions being taken, you'll need additional Systems Manager permissions to execute runbook steps. For more information, see Setting up Automation in the *AWS Systems Manager User Guide*.

To review and execute suggested runbook actions for CloudWatch investigations

1. To view information about a suggested runbook, choose **Review** for information about how to execute the runbook steps.

   On the investigation details page, choose **Suggestions**.
2. In the **Suggestions** pane, review the list of hypotheses based on the system's analysis of the issue under investigation.

   For each hypothesis, you can choose from the following options:

   - **Show reasoning** – View more information about why the system has generated the hypothesis.
   - **View actions** – View the suggested actions for the issue. Not all hypotheses will include suggested actions.
   - **Accept** – Accept the hypothesis and add it to the investigation's **Feed** section.

> **Note**
>
> Accepting the hypothesis doesn't automatically run the associated runbook solution. You can view suggested runbooks before accepting a hypothesis, but you must accept the hypothesis to execute a runbook.

- **Discard** – Reject the hypothesis and don't engage with it any further.

3. After you choose **View action**, in the **Suggested actions** pane, review the list of suggested actions you can take to address the issue. Suggested actions can include one or more of the following:

   - **AWS knowledge articles** – Provides information about steps you can take to manually address the issue, plus a link to more information.
   - **AWS documentation** – Provides links to user documentation topics related to the issue.
   - **AWS-owned runbooks** – Lists one or more Automation runbooks that are managed by AWS that you can run to attempt issue resolution.
   - **Runbooks owned by you** – Lists one or more custom Automation runbooks created by you or someone else in your account or organization, which you can run to attempt issue resolution.

   > **Note**
   >
   > The system automatically generates this list of runbooks by evaluating keywords in your custom runbooks and then comparing them to terms related to the issue being investigated.
   > More keyword matches mean a particular custom runbook appears higher in the **Runbooks owned by you** list.

4. After reviewing the hypothesis, you can examine a specific suggested action further and read related documentation by choosing **Learn more**. You can also choose **Review details** to inspect suggested runbooks owned by AWS and you.

5. When choosing **Review details** for runbooks, do the following:

   a. For **Runbook description**, review the content, which provides an overview of the actions the runbook can take to remediate the issue being investigated. Choose **View steps** to visualize the runbook's workflow and drill into the details of individual steps.

b. For **Input parameters**, specify values for any parameters required by the runbook. These parameters vary from runbook to runbook.

c. For **Execution preview**, carefully review the information. This information explains what the scope and impact would be if you choose to execute the runbook.

   The **Execution preview** content provides the following information:

   - How many accounts and Regions the runbook operation would occur in.
   - The types of actions that would be taken, and how many of each type.

     Action types include the following:

     - `Mutating`: A runbook step would make changes to the targets through actions that create, modify, or delete resources.
     - `Non-Mutating`: A runbook step would retrieve data about resources but not make changes to them. This category generally includes `Describe`, `List`, `Get`, and similar read-only API actions.
     - `Undetermined`: An undetermined step invokes executions performed by another orchestration service like AWS Lambda, AWS Step Functions, or Run Command, a capability of AWS Systems Manager. An undetermined step might also call a third-party API or run a Python or PowerShell script. Systems Manager Automation can't detect what the outcome would be of the orchestration processes or third-party API executions, and therefore can't evaluate them. The results of those steps would have to be manually reviewed to determine their impact.

     For information about supported actions and their impact types, see Remediation impact types of runbook actions in the *AWS Systems Manager User Guide*.

d. Review the preview information carefully before deciding whether to proceed.

   At this point, you can choose one of the following actions:

   - Stop and do not execute the runbook.
   - Change the input parameters before executing the runbook.
   - Execute the runbook with the options you have already selected.

> **Important**
>
> Choosing to execute the runbook incurs charges. For information, see AWS Systems Manager pricing.

6. If you want to execute the runbook, choose **Execute**.

   If you already accepted the hypothesis, the execution runs.

   If you have not already accepted the hypothesis, a dialog box prompts you to accept it before the execution runs.

After you choose **Execute** for a runbook, that action is added to the **Feed** pane of the investigation. From the investigation, you can monitor new data in the metrics in the findings to see if the runbook actions are correcting the issue.

Manage your current investigations

You can view a list of your current investigations, end active investigations, re-open archived investigations, rename, and delete investigations. You can take these actions on individual investigations, or in bulk.

To manage your current investigations

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.
3. (Optional) Filter the investigations displayed in the list by name or investigation state.
4. Select the checkboxes for the investigation or investigations that you want to take action on.
5. Choose **End investigation**, **Rename**, or **Delete**.

   You will be prompted to confirm your action or to input a new investigation title.

Restart an archived investigation

You can restart archived investigations.

To restart an archived investigation

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.
3. Choose the name of an archived investigation.
4. Choose **Restart investigation**.
5. (Optional)Update incident reports.

   Any incident reports generated from the original investigation remain available in the investigation history. You can access these reports from the investigation details page. If the restarted investigation discovers more facts, you can regenerate the incident report using the following steps:

   a. Choose **Incident report** to regenerate your incident report with new or updated facts.
   b. From the **Incident report** page, review updated facts.
   c. Choose **Regenerate** to update your incident report. If the **Regenerate** button is disabled, no new facts are present.

   We recommend that you don't leave investigations open indefinitely, because alarm state transitions related to the investigation will keep being added to the investigation as long as it is open.

Cross-account investigations

Cross-account CloudWatch investigations enables you to investigate application issues that span multiple AWS accounts from a centralized monitoring account. This feature allows you to correlate telemetry data, metrics, and logs across up to 25 accounts, in addition to the monitoring account, to gain comprehensive visibility into distributed applications and troubleshoot complex multi-account scenarios.

Topics

- Prerequisites
- Setup your monitoring account for cross-account access
- Setup your source account(s) for cross-account access
- Investigating multi-account issues

Prerequisites

- Multi-account investigation requires you to already have cross-account observability set up in order to view cross-account telemetries. To complete the prerequisite, set up either cross-account observability or the cross-account dashboard.
- Setup an investigation group. For cross-account observability, this should be in the monitoring account. You can also set them up in the source accounts and run single account investigations there.

Setup your monitoring account for cross-account access

Setup your monitoring account for cross-account access

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Configuration**.
3. Under **Configure Cross-account access**, select **Configure**.
4. Add the Account ID for up to 25 accounts under the **List source accounts** section.
5. Update your IAM role.

   a. Automatically

   - If you choose **Automatically update the assistant role (recommended)** , this creates a customer managed policy named `AIOpsAssistantCrossAccountPolicy-${guid}` that includes the `sts:AssumeRole` statements needed to assume the assistant role in the specified source accounts. Choosing the automatic update option defaults the IAM role name to `AIOps-CrossAccountInvestigationRole` in the source accounts .

     JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::777777777777:role/AIOps-
CrossAccountInvestigationRole",
```

```
            "arn:aws:iam::555555555555:role/AIOps-
CrossAccountInvestigationRole",
            "arn:aws:iam::666666666666:role/AIOps-
CrossAccountInvestigationRole"
        ]
    }
}
```

- If the monitoring account owner removes a source account from the cross-account configuration, the IAM policy will not update automatically. You must manually update the IAM role and policy to make sure it always has the minimum permissions possible.
- You might reach the limit of managed policies per role if the permissions are not manually updated when a source account is removed. You must delete any unused managed policies attached to your investigation role.

b. Manually

- The following example shows the trust policy required for the assistant role:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAIOpsAssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": "aiops.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId": "arn:aws:aiops:us-
east-1:123456789012:investigation-group/AaBbcCDde1EfFG2g"
                }
            }
        }
    ]
}
```

You can use the AWS CLI to create the custom source account role and then attach the `AIOpsAssistantPolicy` to the role using the following commands, replacing the placeholder values with the appropriate values for your environment:

```
aws iam create-role
 --role-name custom-role-name
 --assume-role-policy-document
    '{
       "Version": "2012-10-17",
       "Statement": [
                 {
                      "Effect": "Allow",
                      "Principal": { "AWS": "investigation-group-role-arn"
                          },
                      "Action": "sts:AssumeRole",
                      "Condition": {
                                "StringEquals": {
                                        "sts:ExternalId": "investigation-
group-arn"
                                          } } } ] }'

aws iam attach-role-policy
 --role-name custom-role-name
 --policy-arn arn:aws:iam::aws:policy/AIOpsAssistantPolicy
```

- To grant cross-account access, the permission policy of the assistant role in the monitoring account must contain the following. If you are configuring the monitoring account manually, the role name can be whatever you choose. It does not default to `AIOps-CrossAccountInvestigationRole`, make sure to specify the name of the assistant role for each of the source accounts.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [

 "arn:aws:iam::777777777777:role/custom_source_account_role_name",
```

```
 "arn:aws:iam::555555555555:role/custom_source_account_role_name",

 "arn:aws:iam::666666666666:role/custom_source_account_role_name"
        ]
    }
}
```

- Use the AWS CLI to update the monitoring account investigation group with the custom source account role ARN using the following command, replacing the placeholder values with the appropriate values for your environment:

```
aws aiops update-investigation-group
 --identifier investigation-group-id
 --cross-account-configurations sourceRoleArn=sourceRoleArn1
 sourceRoleArn=sourceRoleArn2
```

For more details on this command, see the AWS CLI Command Reference.

Setup your source account(s) for cross-account access

1. Provision an IAM role with the name `AIOps-CrossAccountInvestigationRole` if you selected the **Automatically update the assistant role** option to set up the monitoring account. If you used the manual setup option, provision the IAM role with your customized source account role name.

   a. Attach the AWS managed policy `AIOpsAssistantPolicy` to the role in the IAM console.

   b. The trust policy of the role on the source account looks like this. `ExternalID` must be specified on the policy. Use the monitoring account investigation group ARN.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "arn:aws:iam::123456789012:role/investigation-
role-name"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId": "investigation-group-arn"
                }
            }
        }
    ]
}
```

2. This must be done in each of the source accounts.
3. If you set up the monitoring account role through the console, the role name of the source account defaults to `AIOps-CrossAccountInvestigationRole`.
4. Confirm access by logging into the monitoring account, navigating to **Investigation Group**, then **Configuration**, and then choosing **Cross-account setup**.

   Make sure the source account shows up in the cross-account configuration, and the status is **Linked to monitoring account**.

Investigating multi-account issues

After you set up CloudWatch cross-account observability dashboard, you can view and investigate from a cross-account telemetry in your monitoring account. You must add a cross-account telemetry from the source account in order to run an investigation into that source account.

For detailed information about how to create an investigation, see Investigate operational issues in your environment.

Generate incident reports

Incident reports help you more quickly and easily write a report about your incident investigation. You can use this report to provide details to management or to help your team learn from the incident and take actions to prevent future such occurrences. The structure of the report is based on industry standards for these types of reports and can be copied into other repositories for long-term retention.

When you use the AWS Management Console to create an *investigation group* resource in CloudWatch investigations, an IAM role is created for the group to give it access to resources during the investigation. Generating CloudWatch investigations incident reports requires additional permissions be granted to your investigation group. The new managed policy `AIOpsAssistantIncidentReportPolicy` provides the required permissions and is automatically added to investigation groups created using the AWS Management Console after October 10, 2025. For more information, see AIOpsAssistantIncidentReportPolicy.

> **Note**
>
> If you are using the CDK or SDK, you must explicitly add the investigation group role and specify the role policy or equivalent inline permissions on the role. For more details about permissions, see Security in CloudWatch investigations

These reports capture investigation findings, root causes, timeline events, and recommended corrective actions in a structured format that can be easily shared with stakeholders and used for organizational learning.

Incident report generation is included at no additional charge for all CloudWatch investigations users and integrates seamlessly with your investigation workflow.

How incident reports work

1. Run an investigation on your incident.
2. Accept at least one hypothesis. Each hypothesis you accept is considered for the report. The hypothesis doesn't need to be 100% accurate.
3. Choose **Incident report**. During the investigation the AI parsed the data collected for your investigation and derived facts. Facts are atomic pieces of information about your incident that form the basis of generating the report. Fact extraction can take a few minutes.
4. When fact extraction is finished, you can review the facts available in the following areas:
   a. **Incident Overview** – High-level overview of the incident including its severity, duration, and operational hypothesis.
   b. **Impact Assessment** – Metrics and analysis related to the impact of the incident on customers, service function, and business operations.
   c. **Detection and Response** – Metrics and analysis related to how and when the incident was detected and how you responded to the incident.

d. **Root Cause Analysis** – Detailed analysis of underlying causes based on investigation hypotheses.

e. **Mitigation and Resolution** – Metrics and analysis related to mitigation steps and resolution measures, along with the time measurement for incident mitigation and resolution.

f. **Learning and Next Steps** – A list of recommended actions for your team to consider, automatically generated from the investigation findings. These recommendations may include preventive measures against similar incidents, as well as suggested improvements to your monitoring and response processes.

5. After reviewing the facts, choose **Generate report** to create a comprehensive analysis of the incident. While the selected facts serve as key reference points, the report draws from all available information gathered during the investigation. This process can take a few minutes.
6. After generating the report, you can then either:
   - Use the report as is:
     - Copy it to edit in your external editor if needed
     - Save it for later reference
   - Enhance the report by adding more data:
     - Choose **Add facts** (recommended method) to input additional text-based content such as incident tickets or custom narratives. The AI will analyze this content to augment existing facts or infer new ones.
     - Edit facts directly (use sparingly) - Manually edited facts may create inconsistencies with the investigation timeline. This should be used only as a last resort when **Add facts** doesn't achieve the desired result.
   - Choose **Regenerate report** to produce a new report using the updated information.

Topics

- Understanding AI-derived facts in incident reports
- Incident report terminology
- Generate a report from an investigation
- Using 5 Whys analysis in incident reports

Understanding AI-derived facts in incident reports

AI-derived facts form the foundation of CloudWatch investigations incident reports, representing information that the AI system considers objectively true or highly probable based on comprehensive analysis of your AWS environment. These facts emerge through a sophisticated process that combines machine learning pattern recognition with systematic verification methods, creating a robust framework for incident analysis that maintains the operational rigor required for production environments.

Understanding how AI-derived facts are developed helps you evaluate their reliability and make informed decisions during incident response. The process represents a hybrid approach where artificial intelligence augments human expertise rather than replacing it, ensuring that the insights generated are both comprehensive and trustworthy.

The development process of AI-derived facts

The journey from raw telemetry data to actionable AI-derived facts begins with pattern observation, where the CloudWatch investigations AI analyzes vast amounts of AWS telemetry using sophisticated machine learning algorithms. The AI examines your CloudWatch metrics, logs, and traces across multiple dimensions simultaneously, identifying recurring patterns and relationships that might not be immediately apparent to human operators. The analysis encompasses temporal patterns that reveal when incidents typically occur and their duration characteristics, service correlations that show how different AWS services interact during failure scenarios, metric anomalies that precede or accompany incidents, and log event sequences that indicate specific failure modes.

Consider, for example, how the AI might observe that in your environment, Amazon EC2 instance CPU utilization consistently spikes to above 90% approximately 15 minutes before application response times exceed acceptable thresholds. This temporal relationship, when observed across multiple incidents, becomes a significant pattern worthy of further investigation. The AI doesn't simply note the correlation; it measures the statistical significance of the relationship and considers various confounding factors that might influence the pattern.

From these observed patterns, the AI moves into hypothesis generation, formulating potential explanations for the relationships it has discovered. This process involves creating multiple competing hypotheses and ranking them by probability based on the strength of supporting evidence. When the AI observes that CPU spikes precede response time degradation, it might generate several hypotheses: resource exhaustion due to insufficient compute capacity, memory

leaks causing increased CPU overhead, or inefficient algorithms triggered by specific input patterns. Each hypothesis receives a preliminary confidence level based on how well it explains the observed data and aligns with known AWS service behaviors.

The human verification and validation of these hypotheses ensures that these AI-generated insights meet operational standards before becoming facts in your incident reports. This process involves correlating AI-derived patterns with established AWS service behavior models, checking consistency with industry best practices for incident response, and validating against historical incident data from similar environments. The AI must demonstrate that its findings are reproducible across different analysis methods and time periods, meet statistical significance requirements for operational decision-making, align with empirical observations of AWS service behavior, and provide actionable insights for incident resolution or prevention.

Throughout this process, the AI faces several inherent challenges that you should understand when interpreting AI-derived facts. The distinction between correlation and causation remains a fundamental challenge; while the AI might identify strong correlations between network traffic spikes and incident occurrence, establishing direct causation requires additional investigation and domain expertise. Hidden variables that exist outside the scope of AWS telemetry, such as third-party service dependencies or external network provider issues, may influence incidents without being captured in the AI analysis. The quality of AI-derived facts depends entirely on the completeness and accuracy of the underlying CloudWatch data, making comprehensive monitoring coverage essential for reliable insights.

Novel incident patterns present another challenge, as those are not present in AI training data, and AIs often struggle to interpret unfamiliar failure modes. This limitation underscores the importance of human expertise in interpreting AI-derived facts and supplementing them with domain knowledge and contextual understanding.

Applying AI-derived facts in incident response

AI excels at identifying patterns across large datasets that would be impractical for humans to analyze manually, providing insights that can significantly accelerate incident diagnosis and resolution. AI works best when combined with human expertise that can provide context, validate conclusions, and identify factors that may not be captured in telemetry data.

The most effective approach involves treating AI-derived facts as highly informed starting points for investigation rather than definitive conclusions. When the AI identifies a fact such as "Database connection pool exhaustion preceded the incident by 8 minutes," this provides a valuable lead that

can be quickly verified through targeted analysis of database metrics and application logs. The fact gives you a specific timeframe and potential root cause to investigate, dramatically reducing the time needed to identify the issue compared to manually searching through all available telemetry.

Data quality plays a crucial role in the reliability of AI-derived facts. Comprehensive CloudWatch monitoring coverage provides the AI access to complete and accurate information for analysis. Gaps in monitoring can lead to incomplete or misleading facts, as the AI can only work with the data available to it. Organizations that use thorough observability practices that include detailed metrics collection, comprehensive logging, and distributed tracing are more likely to have accurate and actionable AI-derived facts in their incident reports.

Incident report terminology

The following terms are used in CloudWatch investigations incident reports:

AI-derived fact

: A piece of information or observation that the AI system considers to be objectively true or highly probable based on the available data, telemetry, logs, and historical patterns within AWS services. These facts are derived through algorithmic analysis and machine learning models, and while they are treated as reliable by the system, they should be subject to human verification, especially in critical decision-making contexts. AI-derived facts may include correlations between events, anomaly detections, or inferences about system behavior that might not be immediately apparent to human operators.

Corrective actions

: Specific, actionable steps recommended by CloudWatch investigations to address the root cause of an incident and prevent its recurrence, based on AWS best practices and the specific context of the affected resources.

Fact categories

: Structured groupings of incident-related information, such as impact metrics, detection details, and mitigation steps, used to organize data for report generation.

Impact assessment

: A quantitative and qualitative evaluation of an incident's effects on system performance, user experience, and business operations, derived from CloudWatch metrics and other AWS service data added to the investigation.

Incident report generation

An automated process that creates comprehensive documentation of an operational incident, including its timeline, impact, root cause, and resolution steps, based on data collected during a CloudWatch investigations investigation.

Investigation Feed

A chronological display of accepted observations, hypotheses, and user-added notes within a CloudWatch investigations investigation, serving as the primary record of the investigation's progress and findings.

Lessons learned

Automatically generated insights and improvement opportunities identified through the incident investigation process, aimed at enhancing system reliability, operational efficiency, and incident response capabilities across the organization.

Report assessment

An automated evaluation of the generated incident report, identifying potential data gaps or areas requiring additional information to improve report completeness and quality.

Root cause analysis

A systematic process of identifying the fundamental reason for an operational issue, using CloudWatch investigations AI-driven hypotheses and correlations across multiple AWS services.

Suggestions tab

A feature in CloudWatch investigations that presents AI-generated observations and hypotheses about potential causes or related issues, based on analysis of system telemetry and logs.

Timeline events

A chronological sequence of significant occurrences during an incident, automatically extracted from CloudWatch logs, metrics, and other AWS service data to provide a clear overview of incident progression.

Generate a report from an investigation

You can generate incident reports from in-progress or completed investigations. Incident reports generated early in an investigation may not include key facts such as root causes and

recommended actions. When the investigation is active you can edit the facts available to supplement the investigation with additional information. After the investigation is ended, you can't edit or add facts to the investigation.

Prerequisites

Before generating an incident confirm the following requirements are met:

- Make sure the investigation group uses the required KMS key and has appropriate IAM policies attached to its role for decrypting data from AWS services. If your AWS resources are encrypted with customer-managed KMS keys, you must add IAM policy statements to the investigation group role to grant CloudWatch Investigations the permissions needed to decrypt and access this data.
- Investigation group role has been granted the following permissions:
  - `aiops:GetInvestigation`
  - `aiops:ListInvestigationEvents`
  - `aiops:GetInvestigationEvent`
  - `aiops:PutFact`
  - `aiops:UpdateReport`
  - `aiops:CreateReport`
  - `aiops:GetReport`
  - `aiops:ListFacts`
  - `aiops:GetFact`
  - `aiops:GetFactVersions`

> **Note**
>
> You can add these permissions as an inline policy to the investigation group role, or attach an additional permissions policy to investigation group role. For more information see, Permissions for incident report generation.
> The new managed policy `AIOpsAssistantIncidentReportPolicy` provides the required permissions and is automatically added to investigation groups created after October 10, 2025. For more information, see AIOpsAssistantIncidentReportPolicy.

To generate an incident report

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, choose **AI Operations**, **Investigations**.
3. Choose the name of an investigation.
4. On the investigation page, under **Feed** accept any additional relevant hypotheses and add any additional notes to the investigation.

   > **Note**
   >
   > Report generation requires an investigation with at least one accepted hypothesis.

5. On the top of the investigation page, choose **Incident report**. Wait while the relevant facts of the investigation are collected and synced.
6. On the **Incident Report** page review the facts being used to generate the report. The facts are available in the right pane. Navigate through the fact category tab using the left and right arrows, or expand the pane to see all of the categories.

   a. Choose **Edit** on a fact panel to manually add or edit the data in that category.

   b. Choose **View details** on a fact panel to see the supporting evidence and fact history gathered by the AI assistant. You can also choose **Edit** within the fact detail window.

   c. Choose **Add facts** if you want to provide additional context to the investigation, such as external events or extenuating circumstances.

7. Choose **Generate report**.

   CloudWatch investigations will analyze the investigation data and generate a structured report. This process might take some time.

8. Review the generated report in the preview pane. The report will include:

   - Automatically extracted timeline events
   - Root cause analysis based on accepted hypotheses
   - Impact assessment derived from investigation telemetry
   - Recommended corrective actions and lessons learned following AWS best practices

9. To retain a copy of the report in a different location, you can choose to copy the text of the report and paste it into your desired location.

10. Choose **Report assessment** to review a list of data gaps in the report. You can use this information to gather additional data for the report and then update the facts accordingly and regenerate the report.

Using 5 Whys analysis in incident reports

When generating incident reports, CloudWatch investigations can perform a 5 Whys root cause analysis to systematically identify the underlying causes of operational issues. This structured approach enhances your incident reports with deeper insights and actionable remediation steps.

This feature uses Amazon Q to provide a conversational chat. The user signed into the AWS Management Console must have the following permissions:

```
{
    "Sid" : "AmazonQAccess",
    "Effect" : "Allow",
    "Action" : [
       "q:StartConversation",
       "q:SendMessage",
       "q:GetConversation",
       "q:ListConversations",
       "q:UpdateConversation",
       "q:DeleteConversation",
       "q:PassRequest"
     ],
    "Resource" : "*"
 }
```

You can add these permissions directly, or by attaching either the AIOpsConsoleAdminPolicy or AIOpsOperatorAccess managed policy to the user or role.

What is 5 Whys analysis?

The 5 Whys is a root cause analysis technique that asks "why" repeatedly to drill down from incident symptoms to fundamental causes. Each answer becomes the basis for the next question, creating a logical chain that reveals the true root cause rather than just surface-level symptoms.

During incident report generation, CloudWatch investigations uses this method to analyze investigation findings and provide structured root cause analysis that goes beyond immediate technical failures to identify process, configuration, or systemic issues.

Benefits for incident reporting

Including 5 Whys analysis in incident reports provides several advantages:

- **Comprehensive root cause identification** - Moves beyond immediate technical causes to identify underlying process or system issues
- **Actionable remediation plans** - Provides specific, targeted actions to prevent recurrence rather than temporary fixes
- **Organizational learning** - Documents the complete causal chain for future reference and team knowledge sharing
- **Structured analysis** - Ensures systematic investigation rather than ad-hoc problem solving

Example scenarios in incident reports

Database connection failure incident

Initial incident: E-commerce application experiencing widespread 500 errors

1. **Why 1:** Why are users getting 500 errors? The application cannot connect to the primary database.
2. **Why 2:** Why can't the application connect to the database? The database instance ran out of available connections.
3. **Why 3:** Why did the database run out of connections? A batch processing job opened many connections without properly closing them.
4. **Why 4:** Why didn't the batch job close connections properly? The job's error handling doesn't include connection cleanup in failure scenarios.
5. **Why 5:** Why wasn't proper error handling implemented? Code review process doesn't include specific checks for resource management patterns.

Root cause: Inadequate code review standards for resource management

Recommended actions: Update code review checklist, implement connection pooling monitoring, add automated resource leak detection

Performance degradation incident

Initial incident: API response times increased from 200ms to 5000ms during traffic spike

1. **Why 1:** Why did response times increase? CPU utilization reached 100% on all application instances.
2. **Why 2:** Why didn't auto scaling add more instances? Auto scaling was triggered but new instances failed health checks.
3. **Why 3:** Why did new instances fail health checks? The application startup process takes 8 minutes, longer than the health check timeout.
4. **Why 4:** Why does startup take so long? The application downloads large configuration files from S3 on every startup.
5. **Why 5:** Why wasn't this startup delay considered in auto scaling configuration? Performance testing was done with pre-warmed instances, not cold starts.

Root cause: Performance testing methodology doesn't reflect production auto scaling scenarios

Recommended actions: Include cold start testing, optimize application startup, adjust health check timeouts, implement configuration caching

Complex incident with branch analysis

Initial incident: OpenSearch Serverless customers experienced 48.3% availability degradation for 11 hours

Main analysis chain:

1. **Why 1:** Why did customers experience service degradation? Service availability dropped to 48.3% due to incorrect ingester scaling.
2. **Why 2:** Why was ingester scaling incorrect? CortexOperator reduced ingesters from 223 to 174 due to AZ balance miscalculation.
3. **Why 3:** Why did CortexOperator miscalculate AZ balance? The code couldn't process new Kubernetes label formats after version 1.17 upgrade.
4. **Why 4 (Branch A - Technical):** Why didn't the code handle new label formats? The code expected 'failure-domain.beta.kubernetes.io/zone' labels but Kubernetes 1.17 changed to 'topology.kubernetes.io/zone'.
5. **Why 5 (Branch A):** Why wasn't backward compatibility implemented? The label format change wasn't documented in the upgrade notes reviewed during deployment planning.

Branch B - Process Analysis:

1. **Why 4 (Branch B - Process):** Why wasn't this caught in testing? Integration tests used pre-configured clusters with old label formats.
2. **Why 5 (Branch B):** Why didn't testing include label format validation? Test environment setup didn't mirror production Kubernetes version upgrade sequence.

Root causes identified:

- Technical: Missing backward compatibility for Kubernetes label format changes
- Process: Testing methodology doesn't validate version upgrade impacts

Integrated remediation plan: Implement label format detection logic, enhance upgrade testing procedures, add automated compatibility validation, and establish version change impact assessment process.

Using the guided 5 Whys workflow

CloudWatch investigations provides a guided 5 Whys analysis workflow to help you address missing facts and strengthen your incident reports. This feature appears as a suggested workflow when the system identifies opportunities to enhance root cause analysis.

Interactive analysis experience

The 5 Whys analysis in CloudWatch investigations uses an interactive, chat-based approach that guides you through the investigation process. This conversational method helps make sure comprehensive analysis while maintaining logical flow between questions.

Key features of the interactive experience:

- **Fact-based initialization** - The system presents relevant facts from your investigation upfront, using them to pre-populate obvious answers and clearly indicating fact-based versus inference-based suggestions
- **Guided probing** - For each "why" question, the system suggests answers based on available facts, requests specific additional context, and guides you to consider important aspects before proceeding
- **Branch management** - When multiple contributing factors are identified, the system clearly presents branch options, explains relationships between branches, and helps prioritize parallel investigations

- **Progressive validation** - For each response, the system reformulates answers for clarity, seeks confirmation, highlights key insights, and connects findings to broader context

This approach ensures that you capture all relevant information while maintaining focus on the most critical causal relationships.

Accessing the guided workflow:

1. During incident report generation, review the **Facts need attention** section in the right panel.
2. Look for the **Guided 5-Whys analysis** suggestion under **Suggested workflow**.
3. Choose **Guide me** to start the interactive 5 Whys process.
4. Follow the guided prompts to systematically work through each "why" question, building a complete causal chain from symptoms to root cause.

The guided workflow helps make sure you capture comprehensive root cause information by walking you through each step of the 5 Whys methodology. The analysis results are automatically incorporated into your incident report, providing structured documentation for post-incident reviews and organizational learning.

You can also request a 5 Whys analysis through the chat interface by asking questions such as "Perform a 5 Whys analysis for this incident" or "What is the root cause using 5 Whys methodology?"

Handling complex incidents with multiple causes

Some incidents involve multiple contributing factors that require parallel analysis paths. CloudWatch investigations supports branch analysis to make sure all significant causes are identified and addressed.

When branch analysis is needed:

- Multiple independent failures occurred simultaneously
- Different system components contributed to the same customer impact
- Both technical and process failures played significant roles
- Cascading failures created multiple causal chains

Branch analysis process:

1. **Branch identification** - The system identifies points where multiple causes converge or diverge
2. **Parallel investigation** - Each branch is analyzed using the complete 5 Whys methodology
3. **Connection mapping** - Relationships between branches are documented to show how they interact
4. **Integrated resolution** - Remediation plans address all identified root causes and their interactions

This comprehensive approach ensures that complex incidents receive thorough analysis and that all contributing factors are addressed in the final remediation plan.

Best practices for effective 5 Whys analysis

To maximize the effectiveness of 5 Whys analysis in your incident reports, follow these best practices derived from operational experience:

Question formulation guidelines

- **Start with customer impact** - Begin each analysis with the customer-facing problem to maintain focus on business impact
- **Increase technical depth progressively** - Move from business impact to technical details as you progress through the questions
- **Maintain logical continuity** - Make sure each answer naturally leads to the next question without logical gaps
- **Include supporting evidence** - Reference specific metrics, logs, or timeline events to validate each answer

Analysis validation

Validate your 5 Whys analysis using these criteria:

- **Logical flow** - Clear progression from symptoms to root cause with no missing steps
- **Technical accuracy** - Correct terminology, accurate system behavior descriptions, and valid component interactions
- **Completeness** - The analysis explains all observed symptoms and reaches a fundamental cause that, if addressed, would prevent recurrence

- **Actionability** - The root cause identified leads to specific, implementable remediation actions

Common pitfalls to avoid

- **Stopping at symptoms** - Don't conclude the analysis at the first technical failure; continue until you reach systemic or process causes
- **Blame-focused analysis** - Focus on system and process failures rather than individual actions
- **Single-path thinking** - Consider multiple contributing factors and use branch analysis when appropriate
- **Insufficient evidence** - Make sure each answer is supported by concrete data from your investigation

Integration with incident report sections

The 5 Whys analysis integrates with other sections of your incident report to provide comprehensive documentation:

- **Timeline correlation** - Each "why" question can reference specific timeline events, providing temporal context for causal relationships
- **Metrics validation** - Answers are supported by metrics and graphs that demonstrate the technical behaviors described
- **Impact assessment alignment** - The first "why" directly connects to customer impact metrics documented in the impact assessment section
- **Lessons learned foundation** - Root causes identified through 5 Whys analysis directly inform the lessons learned and corrective actions sections

This integration ensures consistency across your incident report and provides stakeholders with a complete, coherent narrative from initial symptoms through root cause to remediation plans.

Integrations with other systems

Topics

- Integration with AWS Systems Manager Automation
- Integration with third-party chat systems

Integration with AWS Systems Manager Automation

CloudWatch investigations is integrated with Automation, a capability of AWS Systems Manager. You don't need to configure integration, but you might need to update AWS Identity and Access Management (IAM) permissions so you can use Automation runbooks.

What is AWS Systems Manager?

Systems Manager helps you centrally view, manage, and operate *managed nodes* at scale in AWS, on-premises, and multicloud environments. In Systems Manager, a managed node is any machine configured for use with Systems Manager. For information, see the *AWS Systems Manager User Guide*.

What is Systems Manager Automation?

Automation performs common maintenance, deployment, troubleshooting, and remediation tasks through the use of *runbooks*. Each runbook defines a number of steps for performing tasks. Each step is associated with a particular *action*. The action determines the inputs, behavior, and outputs of the step. For descriptions of the nearly two dozen actions that are supported for runbooks, see the Systems Manager Automation actions reference in the *AWS Systems Manager User Guide*.

Automation provides over 400 AWS managed runbooks. For details about each runbook, including a step-by-step description of the actions performed when executed, see the Systems Manager Automation runbook reference. Customers can also design their own runbooks to address specific scenarios in their environments. For information, see Creating your own runbooks in the *AWS Systems Manager User Guide*.

For information about working with runbooks in an investigation, see Reviewing and executing suggested runbook remediations for CloudWatch investigations.

Integration with third-party chat systems

By integrating CloudWatch investigations with Amazon Q Developer in chat applications, you can have updates from investigations sent to third-party chat services, including Slack, and Microsoft Teams. The integration is facilitated by Amazon Simple Notification Service.

The following topics describe the required steps in the recommended order:

Topics

- Create and configure the Amazon SNS topic and access policy
- Configure Amazon Q Developer in chat applications
- Add Amazon SNS topics to CloudWatch AI Operations

Create and configure the Amazon SNS topic and access policy

Create an Amazon SNS topic in the same region as your investigation. For more information, see Creating an Amazon Simple Notification Service topic.

To enable CloudWatch investigations to send notifications, you must add an the following access policy to the Amazon SNS topic

```
{
    "Sid": "AIOPSChatPublish",
    "Effect": "Allow",
    "Principal": {
        "Service": "aiops.amazonaws.com"
    },
    "Action": "sns:Publish",
    "Resource": "SNS-TOPIC-ARN",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "account-Id"
        }
    }
}
```

Configure Amazon Q Developer in chat applications

To configure Amazon Q Developer in chat applications for communication with a third-party chat service, use the following tutorials:

- Tutorial: Get started with Slack.
- Tutorial: Get started with Microsoft Teams.

Then, to support using AI assistant actions within chat channels you must provide the Amazon Q Developer in chat applications role with appropriate permissions. When you create a new IAM channel role for the channel, select the **Notifications** and **Amazon Q operations assistant permissions** policy templates.

Attach the **AIOpsOperatorAccess** managed IAM policy to the guardrail policies in Amazon Q Developer in chat applications. This grants permissions to Amazon Q Developer in chat applications to interact with CloudWatch investigations and perform required actions on your behalf.

In the channel configuration, you must also subscribe to the Amazon SNS topic that you created in the previous procedure.

Add Amazon SNS topics to CloudWatch AI Operations

You must use the CloudWatch console to configure CloudWatch investigations to integrate with Amazon SNS. You can do this while you create the investigation group in your account, or later.

If you have already created an investigation group and want to add chat integration, follow these steps.

To add chat integration to an existing investigation group

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **AI Operations**, **Configuration**.
3. On the **Optional configuration** tab, in the **Chat integration** section, do the following:
   - If you have already integrated Amazon Q Developer in chat applications with a third-party chat system, you can choose **Select SNS topic** to choose the Amazon SNS topic to use to send updates to about investigations. This Amazon SNS topic will relay those updates to the chat client.
   - If you want to integrate Amazon Q Developer in chat applications with a third-party chat system, choose **Configure new chat client**. For more information about setting up this configuration, see Getting started with Amazon Q Developer in chat applications.

When incident reports are generated, notifications can be sent to configured chat channels to alert team members that new documentation is available. These notifications include links to the generated reports and can be customized to include key findings or action items.

Security in CloudWatch investigations

This section includes topics about how CloudWatch investigations integrate with AWS security and permissions features.

Topics

- Default CloudWatch investigations permissions, retention, and encryption
- User permissions for your CloudWatch investigations group
- Additional permissions for Database Insights
- How to control what data CloudWatch investigations has access to during investigations
- Encryption of investigation data
- Cross-Region inference

Default CloudWatch investigations permissions, retention, and encryption

When you run investigations using default settings without additional configuration in your account the investigation uses the permissions available to your current console session and only accesses telemetry data using Read-only permissions. No investigation group IAM role configuration or permission policy is needed. However, that means the investigation's access to data is limited by the signed in user's permissions.

This investigation is only available to the same user that started the investigation. The investigation is available to view only for a 24 hour period, after which the investigation is deleted with no recovery option available.

The investigation data is encrypted at rest with an AWS owned key. You can't view or manage AWS owned keys, and you can't use them for other purposes or audit their use. However, you don't have to take any action or change any settings to use these keys. For more information, see AWS KMS keys.

User permissions for your CloudWatch investigations group

AWS has created three managed IAM policies that you can use for your users who will be working with your CloudWatch investigations group.

- AIOpsConsoleAdminPolicy– grants an administrator the ability to set up CloudWatch investigations in the account, access to CloudWatch investigations actions, the management of trusted identity propagation, and the management of integration with IAM Identity Center and organizational access.
- AIOpsOperatorAccess– grants a user access to investigation actions including starting an investigation. It also grants additional permissions that are necessary for accessing investigation events.

- AIOpsReadOnlyAccess– grants read-only permissions for CloudWatch investigations and other related services.

We recommend that you use three IAM principals, granting one of them the **AIOpsConsoleAdminPolicy** IAM policy, granting another the **AIOpsOperatorAccess** policy, and granting the third the **AIOpsReadOnlyAccess** policy. These principals could be either IAM roles (recommended) or IAM users. Then your users who work with CloudWatch investigations would sign on using one of these principals.

Permissions for incident report generation

Incident report generation requires additional permissions to allow the AI to collect events, facts, and then create reports.

Users with **AIOpsConsoleAdminPolicy** can generate, edit, and copy incident reports. By default, your investigation group is assigned the **AIOpsAssistantPolicy** to give it access to resource. However, it does not have the permissions required to generate an investigation report. To give permissions to your investigation group to collate the investigation data into an incident report you must add a policy similar to the following example that includes additional permissions or add the additional actions as an inline policy to the group:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "IncidentReportOperations",
            "Effect": "Allow",
            "Action": [
                "aiops:GetInvestigation",
                "aiops:ListInvestigationEvents",
                "aiops:GetInvestigationEvent",

                "aiops:CreateReport",
                "aiops:UpdateReport",
                "aiops:GetReport",

                "aiops:PutFact",
                "aiops:ListFacts",
```

```
                "aiops:GetFact",
                "aiops:GetFactVersions"
            ],
            "Resource": [
                "arn:aws:aiops:*:*:investigation-group/*"
            ]
        }
    ]
}
```

The new managed policy `AIOpsAssistantIncidentReportPolicy` provides the required permissions and is automatically added to investigation groups created after October 10, 2025. For more information, see AIOpsAssistantIncidentReportPolicy.

Additional permissions for Database Insights

To use Database Insights capabilities during investigations, you must attach the `AmazonRDSPerformanceInsightsFullAccess` managed policy to the IAM role or user that you use to perform investigations. CloudWatch investigations requires these permissions to create and access performance analysis reports for your Amazon RDS database instances.

To attach this policy, use the IAM console to add the `AmazonRDSPerformanceInsightsFullAccess` managed policy to your investigation principal. For more information about this managed policy and its permissions, see AmazonRDSPerformanceInsightsFullAccess.

How to control what data CloudWatch investigations has access to during investigations

When you configure an investigation group in your account, you specify what permissions that CloudWatch investigations has to access your resources during investigations. You do this by assigning an IAM role to the investigation group.

To enable CloudWatch investigations to access resources and be able to make suggestions and hypotheses, the recommended method is to attach the **AIOpsAssistantPolicy** to the investigation group role. This grants the investigation group permissions to analyze your AWS resources during your investigations. For information about the complete contents of this policy, see AIOpsAssistantPolicy.

You can also choose to attach the general AWS **ReadOnlyAccess** to the investigation group role, in addition to attaching **AIOpsAssistantPolicy**. The reason to do this is that AWS updates **ReadOnlyAccess** more frequently with permissions for new AWS services and actions that are released. The **AIOpsAssistantPolicy** will also be updated for new actions, but not as frequently.

If you want to scope down the permissions granted to CloudWatch investigations, you can attach a custom IAM policy to the investigation group IAM role instead of attaching the **AIOpsAssistantPolicy** policy. To do this, start your custom policy with the contents of AIOpsAssistantPolicy and then remove permissions that you don't want to grant to CloudWatch investigations. This will prevent CloudWatch investigations from making suggestions based on the AWS services or actions that you don't grant access to.

> **Note**
>
> Anything that CloudWatch investigations can access can be added to the investigation and seen by your investigation operators. We recommend that you align CloudWatch investigations permissions with the permissions that your investigation group operators have.

Allowing CloudWatch investigations to decrypt encrypted data during investigations

If you encrypt your data in any of the following services with a customer managed key in AWS KMS, and you want CloudWatch investigations to be able to decrypt the data from these services and include them in investigations, you'll need to attach one or more additional IAM policies to the investigation group IAM role.

- AWS Step Functions

The policy statement should include a context key for encryption context to help scope down the permissions. For example, the following policy would enable CloudWatch investigations to decrypt data for a Step Functions state machine.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
        {
            "Sid": "AIOPSKMSAccessForStepFunctions",
            "Effect": "Allow",
            "Principal": {
                "Service": "aiops.amazonaws.com"
            },
            "Action":
            [
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition":
            {
                "StringEquals":
                {
                     "kms:ViaService": "states.*.amazonaws.com",
                     "kms:EncryptionContext:aws:states:stateMachineArn":
 "arn:aws:states:region:accountId:stateMachine:*"
                }
            }
        }
    ]
}
```

For more information about these types of policies and using these context keys, see kms:ViaService and kms:EncryptionContext:*context-key* in the AWS Key Management Service Developer Guide, and aws:SourceArn in the IAM User Guide.

Encryption of investigation data

For the encryption of your investigation data, AWS offers two options:

- **AWS owned keys**– By default, CloudWatch investigations encrypts investigation data at rest with an AWS owned key. You can't view or manage AWS owned keys, and you can't use them for other purposes or audit their use. However, you don't have to take any action or change any settings to use these keys. For more information about AWS owned keys, see AWS owned keys.
- **Customer managed keys**– These are keys that you create and manage yourself. You can choose to use a customer managed key instead of an AWS owned key for your investigation data. For more information about customer managed keys, see Customer managed keys.

Incident reports generated from investigations use the same encryption settings as the parent investigation. This maintains a consistent security posture across your investigation data and documentation

> **Note**
>
> CloudWatch investigations automatically enables encryption at rest using AWS owned keys at no charge. If you use a customer managed key, AWS KMS charges apply. For more information about pricing, see AWS Key Management Service pricing.

For more information about AWS KMS, see AWS Key Management Service.

Using a customer managed key for your investigation group

You can associate an investigation group with a customer managed key, and then all investigations created in that group will use the customer managed key to encrypt your investigation data at rest.

CloudWatch investigations customer managed key usage has the following conditions:

- CloudWatch investigations supports only symmetric encryption AWS KMS keys with the default key spec, `SYMMETRIC_DEFAULT`, and that have usage defined as `ENCRYPT_DECRYPT`.
- For a user to create or update an investigation group with a customer managed key, that user must have the `kms:DescribeKey`, `kms:GenerateDataKey`, and `kms:Decrypt` permissions.
- For a user to create or update an investigation in an investigation group that uses a customer managed key, that user must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions.
- For a user to view investigation data in an investigation group that uses a customer managed key, that user must have the `kms:Decrypt` permission.

Setting up investigations to use a AWS KMS customer managed key

First, if you don't already have a symmetric key that you want to use, create a new key with the following command.

```
aws kms create-key
```

The command output includes the key ID and the Amazon Resource Name (ARN) of the key. You will need those in later steps in this section. The following is an example of this output.

```
{
"KeyMetadata": {
"Origin": "AWS_KMS",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "KeyManager": "CUSTOMER",
        "Enabled": true,
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/6f815f63-e628-448c-8251-
e4EXAMPLE",
        "AWSAccountId": "111122223333",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

Set permissions on the key

Next, set permissions on the key. By default, all AWS KMS keys are private. Only the resource owner can use it to encrypt and decrypt data. However, the resource owner can grant permissions to access the key to other users and resources. With this step, you give the AI Operations service principal permission to use the key. This service principal must be in the same AWS Region where the KMS key is stored.

As a best practice, we recommend that you restrict the use of the KMS key to only those AWS accounts or resources that you specify.

The first step to set the permissions is to save the default policy for your key as `policy.json`. Use the following command to do so. Replace *`key-id`* with the ID of your key.

```
aws kms get-key-policy --key-id key-id --policy-name default --output text > ./
policy.json
```

Open the `policy.json` file in a text editor and add the following policy sections into that policy. Separate the existing statement from the new sections with a comma. These new sections use `Condition` sections to enhance the security of the AWS KMS key. For more information, see AWS KMS keys and encryption context.

This policy provides permissions for service principals for the following reasons:

- The `aiops` service needs `GenerateDataKey` permissions to get the data key and use that data key to encrypt your data while it is stored in rest. The `Decrypt` permission is needed to decrypt your data while reading from the data store. The decryption happens when you read the data using `aiops` APIs or when you update the investigation or investigation event. The update operation fetches the existing data after decrypting it, updates the data, and stores the updated data in the data store after encrypting
- The CloudWatch alarms service can create investigations or investigation events. These create operations verify that the caller has access to the AWS KMS key defined for the investigation group. The policy statement gives the `GenerateDataKey` and `Decrypt` permissions to the CloudWatch alarms service to create investigations on behalf of you.

> **Note**
>
> The following policy assumes that you follow the recommendation of using three IAM principals, and granting one of them the **AIOpsConsoleAdminPolicy** IAM policy, granting another the **AIOpsOperatorAccess** policy, and granting the third the **AIOpsReadOnlyAccess** policy. These principals could be either IAM roles (recommended) or IAM users. Then your users who work with CloudWatch investigations would sign on with one of these principals.
> For the following policy, you'll need the ARNs of those three principals.

```
{
    "Sid": "Enable AI Operations Admin for the DescribeKey permissions",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::{account-id}:role/{AIOpsConsoleAdmin}"
    },
    "Action": [
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        }
```

```
    }
},
{
   "Sid": "Enable AI Operations Admin and Operator for the Decrypt and GenerateDataKey
 permissions",
   "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::{account-id}:role/{AIOpsConsoleAdmin}",
            "arn:aws:iam::{account-id}:role/{AIOpsOperator}"
         ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
       "StringEquals": {
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        },
        "ArnLike": {
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
        }
    }
 },
 {
   "Sid": "Enable AI Operations ReadOnly for the Decrypt permission",
   "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::{account-id}:role/{AIOpsReadOnly}"
    },
    "Action": [
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
       "StringEquals": {
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        },
        "ArnLike": {
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
```

```
        }
    }
 },
 {
   "Sid": "Enable the AI Operations service to have the DescribeKey permission",
   "Effect": "Allow",
    "Principal": {
        "Service": "aiops.amazonaws.com"
    },
    "Action": [
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
       "StringEquals": {
            "aws:SourceAccount": "{account-id}"
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:aiops:{region}:{account-id}:investigation-group/
*"
        }
    }
 },
 {
   "Sid": "Enable the AI Operations service to have the Decrypt and GenerateDataKey
 permissions",
   "Effect": "Allow",
    "Principal": {
        "Service": "aiops.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
       "StringEquals": {
            "aws:SourceAccount": "{account-id}"
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:aiops:{region}:{account-id}:investigation-group/
*"
        },
        "ArnLike": {
```

```
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
        }
    }
 },
 {
    "Sid": "Enable CloudWatch to have the Decrypt and GenerateDataKey permissions",
    "Effect": "Allow",
    "Principal": {
        "Service": "aiops.alarms.cloudwatch.amazonaws.com"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
        "ArnLike": {
            "kms:EncryptionContext:aws:aiops:investigation-group-arn": "arn:aws:aiops:
{region}:{account-id}:investigation-group/*"
        },
        "StringEquals": {
            "aws:SourceAccount": "{account-id}",
            "kms:ViaService": "aiops.{region}.amazonaws.com"
        },
        "StringLike": {
            "aws:SourceArn": "arn:aws:cloudwatch:{region}:{account-id}:alarm:*"
        }
    }
  }
```

After you've updated the policy, assign it to the key by entering the following command.

```
aws kms put-key-policy --key-id key-id --policy-name default --policy file://
policy.json
```

Associate the key with the investigation group

When you use the CloudWatch console to create an investigation group, you can choose to associate the AWS KMS key with the investigation group. For more information, see Set up an investigation group.

You can also associate a customer managed key with an existing investigation group.

Changing your encryption configuration

You can update an investigation group to change between using a customer managed key or a service owned key. You can also change from using one customer managed key to using another. When you make such a change, the change applies to new investigations created after the change. Previous investigations are still associated with the old encryption configuration. Current ongoing investigations also continue using the original key for new data.

As long as a previously-used key is active and Amazon Q has access to it for investigations, you can retrieve the older investigations encrypted with that method, as well as data in current investigations that was encrypted with the previous key. If you delete a previously-used key or revoke access to it, the investigation data encrypted with that key can't be retrieved.

Cross-Region inference

CloudWatch investigations uses *cross-Region inference* to distribute traffic across different AWS Region. Although the data remains stored only in the primary Region, when using cross-Region inference, your investigation data might move outside of your primary Region. All data will be transmitted encrypted across Amazon's secure network.

For details about where cross-Region inference distribution occurs for each Region, see the following table.

Supported CloudWatch investigations geography	Investigation Region	Possible inference Regions
United States (US)	US East (N. Virginia)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	US East (Ohio)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	US West (Oregon)	US East (N. Virginia), US East (Ohio), US West (Oregon)
Europe (EU)	Europe (Frankfurt)	Europe (Frankfurt), Europe (Ireland), Europe (Paris), Europe (Stockholm)

Supported CloudWatch investigations geography	Investigation Region	Possible inference Regions
	Europe (Ireland)	Europe (Frankfurt), Europe (Ireland), Europe (Paris), Europe (Stockholm)
	Europe (Spain)	Europe (Frankfurt), Europe (Ireland), Europe (Paris), Europe (Stockholm)
	Europe (Stockholm)	Europe (Frankfurt), Europe (Ireland), Europe (Paris), Europe (Stockholm)
Asia-Pacific (AP)	Asia Pacific (Hong Kong)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Mumbai)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Singapore)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Sydney)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Tokyo)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Malaysia)	US East (N. Virginia), US East (Ohio), US West (Oregon)
	Asia Pacific (Thailand)	US East (N. Virginia), US East (Ohio), US West (Oregon)

CloudWatch investigations data retention

The retention period that you set for an investigation group determines how long that investigation data is kept. Valid values are seven days to 90 days.

After you first create an investigation, if you don't end it manually, it moves to a CLOSED state automatically after seven days. Then, the retention period determines how long the data is kept after the investigation moves to the CLOSED state. The data that is kept during the retention period includes the data in the investigation, accepted and discarded findings, and AI assistant audit log messages.

When this retention period expires, the investigation data is deleted.

If you manually end an investigation, that also moves the investigation to the CLOSED state and the retention period time begins to be in effect.

> **Note**
>
> Investigations conducted without configuring your CloudWatch investigations settings are linked to individual user sessions and are deleted after 24 hours, with no recovery option available.

Incident report retention

Incident reports generated from investigations follow the same retention policy as their parent investigations.

We recommend copying important incident reports to external systems if you need to retain them beyond the investigation retention period.

For more information, see Generate incident reports.

Troubleshooting

Topics

- CloudWatch investigations cannot assume the necessary IAM roles or permissions. Please verify required roles and permissions are correctly configured

- Unable to identify event source. Verify that the resource exists in your application topology and the resource type is supported.
- Analysis complete. Submit additional findings to receive updated suggestions
- Source account status shows "Pending link to monitoring account"
- Incident report generation issues

CloudWatch investigations cannot assume the necessary IAM roles or permissions. Please verify required roles and permissions are correctly configured

CloudWatch investigations use an IAM role to be able to access information in your topology. This IAM role must be configured with adequate permissions. For more information about the necessary permissions, see How to control what data CloudWatch investigations has access to during investigations.

Unable to identify event source. Verify that the resource exists in your application topology and the resource type is supported.

There are several AWS services and features that we recommend you to enable to provide additional valuable information to CloudWatch investigations. These services and features can help CloudWatch investigations identify event sources. For more information, see (Recommended) Best practices to enhance investigations.

Analysis complete. Submit additional findings to receive updated suggestions

When you see this message, CloudWatch investigations has finished analyzing your topology and telemetry based on the findings that it has found so far. If you think that the root cause hasn't been found, you can manually add more telemetry to the investigation, and this might cause CloudWatch investigations to scan your system again based on the new information.

To add new telemetry, navigate to that service's console and add the telemetry to the investigation. For example, to add a Lambda metric to the investigation, you can do the following:

1. Open the Lambda console.
2. In the **Monitor** section, find the metric.

3. Open the vertical ellipsis context menu

   for the metric, choose **Investigate**, **Add to investigation** Then, in the **Investigate** pane, select the name of the investigation.

Source account status shows "Pending link to monitoring account"

Check that both your monitoring account and source account are set up correctly.

1. Check that the source account ID and the source account role name are correct.
2. The monitoring account role needs to have `sts:AssumeRole` permission to assume the source account role.
3. The source account role needs to have a trust policy for the monitoring account role to assume.
4. The trust policy in the source account role must be properly scoped to have the Investigation Group arn in the `sts:ExternalId` context key:

```
"Condition": {
    "StringEquals": {
        "sts:ExternalId": "investigation-group-arn"
    }
}
```

Incident report generation issues

This section describes common issues you might encounter when generating incident reports and how to resolve them.

Report generation fails or produces incomplete content

If incident report generation fails, verify the following:

- The investigation has at least one accepted hypothesis in the **Feed**
- Your user and your investigation group have the necessary permissions, see User permissions for your CloudWatch investigations group.

Network Monitoring

The topics in this section describe CloudWatch network and internet monitoring capabilities provided by Network Flow Monitor, Internet Monitor and Network Synthetic Monitor. These services help you to gain operational visibility into the network and internet performance and availability of your applications hosted on AWS.

- Network Flow Monitor provides near real-time visibility into network performance, such as packet loss and latency, for traffic between Amazon EC2 instances, as well as traffic toward other AWS services, such as Amazon S3 and Amazon DynamoDB. Network Flow Monitor works by using data from lightweight software agents that you install to run on your instances. These fully-managed agents gather performance statistics from TCP connections and send them to the Network Flow Monitor backend. By creating monitors for specific agents and then using Network Flow Monitor dashboards, you can quickly visualize packet loss and latency of your network connections, and use attribution information to determine where to focus your troubleshooting efforts to improve your end users' experience.
- Internet Monitor uses the connectivity data that AWS captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. You can see a global view of traffic patterns and health events, and easily drill down into information about events. You can also get alerts for internet health events that affect your application clients. In addition, you can use insights that Internet Monitor provides to explore potential improvements to your client experience, by using Amazon CloudFront or routing through different AWS Regions.
- Network Synthetic Monitor uses fully-managed agents to enable you to track and visualize latency and packet loss for hybrid network connections. To gather measurements and enable Network Synthetic Monitor to create health event alerts for your application, you create probes that are sent from your resources hosted on AWS to on-premises destination IP addresses. You don't need to install additional agents to monitor your network performance. As with Internet Monitor, you can set alerts and thresholds, get information to help you quickly troubleshoot issues, and then take action to improve your end user experience.

Topics

- Using Network Flow Monitor
- Using Internet Monitor
- Using Network Synthetic Monitor

Using Network Flow Monitor

Network Flow Monitor gives you near real-time visibility into network performance for traffic between compute resources (Amazon EC2 and Amazon Elastic Kubernetes Service), traffic to other AWS services (Amazon S3 and Amazon DynamoDB), and traffic to the edge of another AWS Region. Network Flow Monitor collects data from lightweight software agents that you install on your instances. These agents gather performance statistics from TCP connections and send this data to the Network Flow Monitor backend service, which calculates the top contributors for each metric type.

Network Flow Monitor also determines if AWS is the cause of a detected network issue, and reports that information for network flows that you choose to monitor details for.

You can view network performance information for network flows for resources in a single account, or you can configure Network Flow Monitor with AWS Organizations to view performance information for multiple accounts in an organization, by signing in with a management or delegated administrator account.

Network Flow Monitor is intended for network operators and application developers who want near real-time insights into network performance. In the Network Flow Monitor console in CloudWatch, you can see performance data for your resources' network traffic that has been aggregated from agents and grouped into different categories. For example, you can see data for flows between Availability Zones or between VPCs. Then, you can create monitors for specific flows that you want to see more details for or track more closely over time.

Using a monitor, you can quickly visualize packet loss and latency of your network connections over a time frame that you specify. For each monitor, Network Flow Monitor also generates a network health indicator (NHI). The NHI value informs you whether there were AWS network issues for the network flows tracked by your monitor during the time period that you're evaluating. Using the NHI information, you can quickly decide whether to focus troubleshooting efforts on an AWS network issue or network problems originating with your workloads.

To see an example of configuring and using Network Flow Monitor, see the following blog post: Visualizing network performance of your AWS Cloud workloads with Network Flow Monitor.

What is Network Flow Monitor?

Network Flow Monitor is a feature of Amazon CloudWatch Network Monitoring. Network Flow Monitor uses fully-managed agents that you install in your AWS workloads to return performance

and availability metrics about network flows. Using Network Flow Monitor, you can access near real-time metrics, including retransmissions and data transferred, for your actual workloads. You can also identify whether an underlying AWS network issue occurred for the network flows tracked by a monitor, by checking network health indicator (NHI) values.

Key features of Network Flow Monitor

- With Network Flow Monitor, you receive near real-time metrics for the latency and packet-loss experienced by TCP-based traffic within your VPC network, so that you can track and investigate network issues for your workload traffic.
- When your AWS workloads experience network degradation, Network Flow Monitor helps you to determine if the problem is caused by your application workload or the underlying AWS infrastructure. Then, you can quickly focus troubleshooting on the area where the issue is occurring.

How to use Network Flow Monitor

With Network Flow Monitor, you install lightweight agents on your instances, which collect and aggregate performance metrics. Network Flow Monitor agents analyze TCP traffic, and then export performance metrics to the Network Flow Monitor service backend.

Agents gather the following metrics for your workloads: TCP round-trip time (RTT), TCP retransmission timeouts, TCP retransmissions, and data (bytes) transferred. After you install agents on your instances, the agents detect the corresponding workloads that are hosted by the instances. The agents then generate network performance metrics and send the metrics to the Network Flow Monitor backend. Metrics are aggregated into categories such as subnets, Availability Zones, VPCs, and AWS services.

Performance metrics for top contributors (by metric type) from all network flows that are in your Network Flow Monitor scope are shown on the **Workload insights** tab in the AWS Management Console. By reviewing the tables and graphs of top contributors, you can determine where there might be impairments that you want to troubleshoot and which workloads you want to monitor on an ongoing basis, by creating a monitor.

With a monitor, you can monitor specific workloads more closely over time and see detailed information about specific network flows. In addition to viewing performance metrics for the top contributors for the network flows that you've selected, you can view network path information with the network hops that a network flow has traversed, to help you troubleshoot issues. In

addition, Network Flow Monitor generates a network health indicator (NHI) for monitors. An NHI value of **Degraded** indicates that there were AWS network issues for at least one of the network flows tracked by your monitor, during the time period that you've selected.

In addition to reviewing the information in monitors that you create, we recommend that you also check back regularly to review metrics on the **Workload insights** page, to see the latest top contributors for performance metrics for your network flows. As you review the latest information, consider if it would be helpful to add or remove workloads from your current monitors, or create new monitors.

Contents

- Supported AWS Regions for Network Flow Monitor
- Components and features of Network Flow Monitor
- How it works
- Pricing for Network Flow Monitor

Supported AWS Regions for Network Flow Monitor

Network Flow Monitor is currently available in the following AWS Regions:

Region name	Region
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1

Region name	Region
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2

Components and features of Network Flow Monitor

Network Flow Monitor uses or references the following concepts.

Agents

An *agent* in Network Flow Monitor is a software application that you install on your AWS compute resources (Amazon EC2 and Amazon EKS). The application has two parts:

- The first part receives events related to TCP connections and is registered within the Linux kernel using eBPF. eBPF is the Linux extended Berkley Packet Filter (eBPF) capability that allows a designated program to receive certain events raised by the Linux kernel.
- The second part aggregates the statistics collected by the eBPF portion. The agent sends the aggregated metrics to the Network Flow Monitor backend about every 30 seconds, with a 5 second potential jitter (in other words, 25 to 35 seconds).

For more information about agents, see How it works.

Top contributors

Top contributors are the network flows that have the highest values for a specific metric (such as retransmissions) in your Network Flow Monitor scope or among the network flows you're tracking in a monitor. Reviewing the flows with the highest reported numbers for performance

metric measurements can help you see where there might be impairments to investigate. Network Flow Monitor returns performance metrics for top contributors in your monitoring scope for *workload insights*. In addition, if you create a monitor, Network Flow Monitor returns performance metrics for top contributors for the network flows that you choose for the monitor.

Local and remote resources

A *local resource* is the host location—or a location of multiple hosts—where a Network Flow Monitor agent is installed. It can be a subnet, a VPC, an Availability Zone, an Amazon EKS cluster, or an AWS Region. For example, consider a workload that consists of an interaction between a web service and a backend database, for example, DynamoDB. In this scenario, the local resource is the subnet of the EC2 instance hosting the web service, which also runs the agent. A network flow is typically directional, though it can be configured to be bi-directional.

A *remote resource* is the other end of a network flow. In this example of a web service with a backend database, DynamoDB is the remote resource. A remote resource can be a subnet, a VPC, an Availability Zone, an AWS service, or an AWS Region. If you specify a Region as a remote resource, Network Flow Monitor measures performance for the network flow up to the edge of the Region. It does not measure performance to specific endpoints within the Region.

A resource is identified by the ARN of the resource, the name of the AWS service, or, for an Availability Zone or Region, by the name of the zone or Region.

Workload insights

Workload insights includes the performance metrics returned for all the network flows in your scope. In the AWS Management Console, the **Workload insights** page provides performance data about workloads where you've installed Network Flow Monitor agents on workload instances. The **Workload insights** page provides a view into your applications that includes the amount of data transferred and several other metrics, grouped into categories of workloads. For example, you can see all the metrics for workloads with traffic between Availability Zones (AZs) or within an AZ. By using these insights, you can select workloads for which you want to create a monitor to see more details and to track network performance on an ongoing basis.

Monitors

You create a *monitor* so that you can monitor, on an ongoing basis, the network performance for one or several specific workloads, and see more detailed information about the network flows. For each monitor, Network Flow Monitor publishes end-to-end performance metrics, and a network health indicator (NHI), which you can use to help determine attribution for

impairments. We recommend that you review information on the **Workload insights** page to see which network flows you want to focus on, and then create a monitor for those flows. Then, by regularly reviewing **Workload insights**, you can decide if you have the monitors that you need, or if creating new monitors would be helpful.

Network health indicator (NHI)

The *network health indicator* (NHI) is a binary value that informs you whether there were AWS network issues for one or more of the network flows tracked by a monitor, during a time period that you choose. When the NHI value is 100, or **Degraded**, there was an AWS network issue for at least one network flow. With the NHI indicator, you can quickly decide whether to focus troubleshooting efforts on an AWS network issue or network problems originating with your workloads.

For more information, see View Network Flow Monitor metrics in CloudWatch.

Scope

In Network Flow Monitor, the *scope* is the account or accounts that you have observability for when you look at network performance indicators. If you sign in as a management account and configure AWS Organizations with CloudWatch, you can set your scope to more than one account in your organization (up to 100 accounts total). Otherwise, if you sign in with an AWS account that does not have management permissions in Organizations, or if you have not configured Organizations with CloudWatch, Network Flow Monitor sets your scope to the account that you're signed in with.

After you have configured Organizations, you can change your scope by adding or removing accounts. However, whenever you change the scope, Network Flow Monitor must create a new topology of the resources in the scope. For more information, see Add multiple accounts to your scope.

Network Flow Monitor generates a unique **scope ID** for the scope. Queries for metrics data use the scope ID to determine the resources that the Network Flow Monitor generates metrics for. (You must install agents to gather and submit metrics data before you can view the performance metrics for an account with Network Flow Monitor.)

Query ID

Network Flow Monitor generates a unique *query ID* for each query that is created to retrieve performance metrics data, such as a query for top contributors for a monitor. By using a query ID with an API call in Network Flow Monitor, you can check the status of a query, stop a query, run the query again, or work with the query in other ways.

Performance metrics

Network Flow Monitor gathers and calculates end-to-end *performance metrics*, including TCP round-trip time (RTT), TCP retransmissions, TCP retransmission time outs, and bytes transferred for each flow that is in your Network Flow Monitor scope. The service aggregates these metrics and returns them to the service backend. You can view top contributors by metric type. When you see an anomaly in Network Flow Monitor, you can also check the network health indicator (NHI) to see if there is an underlying AWS network issue.

Be aware that RTT data can be sparse because RTT is not always calculated.

You can also use Amazon CloudWatch features to create dashboards, alarms, and notifications based on these metrics. For example, you can learn about setting up alarms with Network Flow Monitor metrics by reviewing the information in Create alarms with Network Flow Monitor.

How it works

This section provides information about several aspects of how Network Flow Monitor works.

How Network Flow Monitor agents gather statistics

Agents in Network Flow Monitor are installed on AWS compute resources (Amazon EC2 and Amazon EKS), where they gather performance metrics and send them to the Network Flow Monitor backend. Agents do not have access to the payload of your TCP connections. Agents receive only what is called the "bpf_sock_ops" structure from the Linux kernel. This structure provides the local and remote IP address and the source and destination TCP port, as well as counters and round-trip times. For list of the TCP statistics collected and published by the agent, see View Network Flow Monitor metrics in CloudWatch.

The agent uses the Network Flow Monitor `Publish` API to send metrics to the Network Flow Monitor backend server.

Note: Network Flow Monitor supports up to approximately 5 million flows per minute. This is approximately 5,000 instances (Amazon EC2 and Amazon EKS nodes) with NFM agent installed. Installing agents on more than 5000 instances may affect monitoring performance until additional capacity is available.

How network flows are categorized in Network Flow Monitor

Network Flow Monitor categorizes network flows into classifications depending on where the flows originate and terminate.

Pricing for Network Flow Monitor

With Network Flow Monitor, there are no upfront costs or long-term commitments. Pricing for Network Flow Monitor has two components: resources monitored (agents installed and actively sending data) and CloudWatch metrics vended. Note that you are also charged standard CloudWatch prices for any additional metrics, dashboards, alarms, or insights that you create.

For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the Amazon CloudWatch pricing page.

Get started with Network Flow Monitor

To help you get started, the section provides a high level overview of the steps to configure, and then gain insights, with Network Flow Monitor. For details, see the additional sections in this guide about initializing Network Flow Monitor, deploying agents, and creating monitors.

- Initialize Network Flow Monitor, to accept service-linked role permissions, create a *scope* for monitoring in Network Flow Monitor, and create an initial topology. If you want to observe network performance for network flows for instances in multiple accounts, you must integrate with AWS Organizations, and then add the accounts to your scope. To learn more, see Initialize Network Flow Monitor.
- Deploy *agents* on your instances, by using AWS Systems Manager or by configuring Kubernetes, depending on how your resources are deployed. If you install agents on VPC EC2 instances, make sure that you enable permissions for agents on each instance to send metrics to the Network Flow Monitor backend. To learn more, see Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances.
- Review top contributor metrics for network flows returned by the agents, to gain *workload insights*. Workload insights provide a high-level view of the performance for network flows in the scope you're monitoring.
- Based on the network flows that you want to see detailed network information about, create one or more *monitors*. For each monitor, specify the local and remote resources that you want to monitor network flows between. Using a monitor, you can see detailed metrics and information, including the network health indicator, as well as view network paths for specific network flows, over time periods that you select.
- On a regular basis:
  - Review network flow information in the monitors that you've created, to learn about and help troubleshoot network impairments in your workloads.

- Review workload insights for the network flows that you're monitoring, to determine if the monitors that you've created are covering the most relevant network flows or if it would be helpful to create new monitors.

Network Flow Monitor API operations

The following table lists Network Flow Monitor API operations that you can use with Network Flow Monitor. The table also includes links to relevant documentation.

Action	API operation	More information
Create a flow monitor.	See CreateMonitor	See Create a monitor in Network Flow Monitor
Generate metrics for a scope of resources.	See CreateScope	See Evaluate network flows with workload insights
Remove a monitor.	See DeleteMonitor	See Delete a monitor in Network Flow Monitor
Delete a defined scope.	See DeleteScope	See Delete a monitor in Network Flow Monitor
Get information about a monitor.	See GetMonitor	See Monitor and analyze network flows with a Network Flow Monitor monitor
Get query data for the top contributors in a specific monitor.	See GetQueryResultsMonitorTopContributors	See Monitor and analyze network flows with a Network Flow Monitor monitor
Query the top contributors for a defined scope for workload insights.	See GetQueryResultsWorkloadInsightsTopContributors	See Evaluate network flows with workload insights
Query workload insights data for the top contributors in a specific scope.	See GetQueryResultsWorkloadInsightsTopContributorsData	See Evaluate network flows with workload insights

Action	API operation	More information
Check the status of a query for top contributors in a monitor to make sure that it has succeeded before reviewing the results.	See GetQueryStatusMonitorTopContributors	N/A
Check the status of a query on workload insights for top contributors to make sure that it has succeeded before reviewing the results.	See GetQueryStatusWorkloadInsightsTopContributors	N/A
Check the status of a query on workload insights data for top contributors to make sure that it has succeeded before reviewing the results.	See GetQueryStatusWorkloadInsightsTopContributorsData	N/A
Get information about an account, or scope, including name, status, tags, and target details.	See GetScope	See Monitor and analyze network flows with a Network Flow Monitor monitor
List all monitors in an account.	See ListMonitors	Monitor and analyze network flows with a Network Flow Monitor monitor
List all scopes for an account.	See ListScopes	N/A
List all tags for a specific resource.	See ListTagsForResource	See Monitor and analyze network flows with a Network Flow Monitor monitor
See query data for top contributors in a monitor.	See StartQueryMonitorTopContributors	See Monitor and analyze network flows with a Network Flow Monitor monitor

Action	API operation	More information
Start a query to gather workload insights data for top contributors.	See StartQueryWorkloadInsightsTopContributors	See Evaluate network flows with workload insights
Stop a query for top contributors in a monitor.	See StopQueryMonitorTopContributors	N/A
Stop a query on workload insights data for top contributors.	See StopQueryWorkloadInsightsTopContributors	N/A
Stop a query on workload insights data for top contributors.	See StopQueryWorkloadInsightsTopContributorsData	N/A
Add a tag to a resource.	See TagResource	See Edit a monitor in Network Flow Monitor
Remove a tag from a resource.	See UntagResource	See Edit a monitor in Network Flow Monitor
Update a monitor to add or remove local or remote resources.	See UpdateMonitor	See Edit a monitor in Network Flow Monitor
Modify a scope to add or remove resources that Network Flow Monitor will generate metrics on.	See UpdateScope	N/A

Examples of using the CLI with Network Flow Monitor

This section includes examples for using the AWS Command Line Interface with Network Flow Monitor operations.

Before you begin, make sure that you log in to use the AWS CLI with the AWS account that provides the scope that you want to use to monitor network flows. For more information about using API actions with Network Flow Monitor, see the Network Flow Monitor API Reference Guide.

Topics

- Create a monitor
- View monitor details
- Create a scope
- Delete a monitor
- Delete a scope
- Get information about a monitor
- Retrieve data on a specific queries
- See scope information
- See a list of monitors for an account
- See a list of scopes for an account
- See the list of tags for a monitor
- Starting and stopping queries
- Tag a monitor
- Remove a tag from a monitor
- Update an existing monitor

Create a monitor

To create a monitor with the AWS CLI, use the `create-monitor` command. The following example creates a monitor named demo in the specified account.

```
aws networkflowmonitor create-monitor \
        --monitor-name demo \
        --local-resources type="AWS::EC2::VPC",identifier="arn:aws:ec2:us-
east-1:111122223333:vpc/vpc-11223344556677889"  \
        --scope-arn arn:aws:networkflowmonitor:us-east-1:111122223333:scope/sample-
aaaa-bbbb-cccc-44556677889
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/demo",
        "monitorName": "demo",
        "monitorStatus": "ACTIVE",
        "tags": {}
    }
```

For more information, see Create a monitor in Network Flow Monitor.

View monitor details

To view information about a monitor with the AWS CLI, use the `get-monitor` command.

```
aws networkflowmonitor get-monitor --monitor-name "TestMonitor"
```

Output:

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/
TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
}
```

Create a scope

The following `create-scope` example creates a scope that is the set of resources for which Network Flow Monitor will generate network traffic metrics.

```
aws networkflowmonitor create-scope \
        --targets '[{"targetIdentifier":{"targetId":
{"accountId":"111122223333"},"targetType":"ACCOUNT"},"region":"us-east-1"}]'
```

Output:

```
    {
        "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
        "status": "IN_PROGRESS",
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

Delete a monitor

The following `delete-monitor` example deletes a monitor named `Demo` in your account.

```
aws networkflowmonitor delete-monitor \
        --monitor-name Demo
```

This command produces no output.

For more information, see Delete a monitor in Network Flow Monitor.

Delete a scope

The following `delete-scope` example deletes the specified scope.

```
aws networkflowmonitor delete-scope \
        --scope-id sample-aaaa-bbbb-cccc-44556677889
```

This command produces no output.

For more information, see Components and features of Network Flow Monitor.

Get information about a monitor

The following `get-monitor` example displays information about the monitor named `demo` in the specified account.

```
aws networkflowmonitor get-monitor \
```

```
        --monitor-name Demo
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo",
        "monitorName": "Demo",
        "monitorStatus": "ACTIVE",
        "localResources": [
            {
                "type": "AWS::EC2::VPC",
                "identifier": "arn:aws:ec2:us-east-1:111122223333:vpc/
vpc-11223344556677889"
            }
        ],
        "remoteResources": [],
        "createdAt": "2024-12-09T12:21:51.616000-06:00",
        "modifiedAt": "2024-12-09T12:21:55.412000-06:00",
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

Retrieve data on a specific queries

The following sections provide example CLI commands to retrieve query statuses.

get-query-results-workload-insights-top-contributors-data

The `get-query-results-workload-insights-top-contributors-data` example returns the data for the specified query.

```
aws networkflowmonitor get-query-results-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "datapoints": [
            {
```

```
                "timestamps": [
                    "2024-12-09T19:00:00+00:00",
                    "2024-12-09T19:05:00+00:00",
                    "2024-12-09T19:10:00+00:00"
                ],
                "values": [
                    259943.0,
                    194856.0,
                    216432.0
                ],
                "label": "use1-az6"
            }
        ],
        "unit": "Bytes"
    }
```

get-query-results-workload-insights-top-contributors

The following `get-query-results-workload-insights-top-contributors` example returns the data for the specified query.

```
aws networkflowmonitor get-query-results-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "topContributors": [
            {
                "accountId": "111122223333",
                "localSubnetId": "subnet-SAMPLE1111",
                "localAz": "use1-az6",
                "localVpcId": "vpc-SAMPLE2222",
                "localRegion": "us-east-1",
                "remoteIdentifier": "",
                "value": 333333,
                "localSubnetArn": "arn:aws:ec2:us-east-1:111122223333:subnet/
subnet-2222444455556666",
                "localVpcArn": "arn:aws:ec2:us-east-1:111122223333:vpc/
vpc-11223344556677889"
            }
        ]
```

```
    }
```

get-query-status-monitor-top-contributors

The following `get-query-status-monitor-top-contributors` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-monitor-top-contributors \
        --monitor-name Demo \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

get-query-status-workload-insights-top-contributors-data

The following `get-query-status-workload-insights-top-contributors-data` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

get-query-results-workload-insights-top-contributors

The following `get-query-results-workload-insights-top-contributors` example displays the current status of the query in the specified account.

```
aws networkflowmonitor get-query-status-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
```

```
        --query-id sample-dddd-eeee-ffff-44556677889
```

Output:

```
{
        "status": "SUCCEEDED"
    }
```

For more information, see Evaluate network flows with workload insights.

See scope information

The following `get-scope` example displays information about a scope, such as status, tags, name, and target details.

```
aws networkflowmonitor get-scope \
        --scope-id sample-aaaa-bbbb-cccc-11112222333
```

Output:

```
{
        "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
        "status": "SUCCEEDED",
        "scopeArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:scope/sample-
aaaa-bbbb-cccc-11112222333",
        "targets": [
            {
                "targetIdentifier": {
                    "targetId": {
                        "accountId": "111122223333"
                    },
                    "targetType": "ACCOUNT"
                },
                "region": "us-east-1"
            }
        ],
        "tags": {}
    }
```

For more information, see Components and features of Network Flow Monitor.

See a list of monitors for an account

The following `list-monitors` example returns all the monitors in the specified account.

```
aws networkflowmonitor list-monitors
```

Output:

```
{
        "monitors": [
            {
                "monitorArn": "arn:aws:networkflowmonitor:us-
east-1:111122223333:monitor/Demo",
                "monitorName": "Demo",
                "monitorStatus": "ACTIVE"
            }
        ]
    }
```

For more information, see Components and features of Network Flow Monitor.

See a list of scopes for an account

The following `list-scopes` example lists all the scopes in the specified account.

```
aws networkflowmonitor list-scopes
```

Output:

```
{
        "scopes": [
            {
                "scopeId": "sample-aaaa-bbbb-cccc-11112222333",
                "status": "SUCCEEDED",
                "scopeArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:scope/
sample-aaaa-bbbb-cccc-11112222333"
            }
        ]
    }
```

For more information, see Components and features of Network Flow Monitor.

See the list of tags for a monitor

The following `list-tags-for-resource` example returns all the tags associated with the specified resource.

```
aws networkflowmonitor list-tags-for-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo
```

Output:

```
{
        "tags": {
            "Value": "Production",
            "Key": "stack"
        }
    }
```

For more information, see Tagging your Amazon CloudWatch resources.

Starting and stopping queries

The following sections provide example CLI commands for starting and stopping queries in Network Flow Monitor.

start-query-monitor-top-contributors

The following `start-query-monitor-top-contributors` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-monitor-top-contributors \
        --monitor-name Demo \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
```

```
    }
```

For more information, see Evaluate network flows with workload insights.

start-query-workload-insights-top-contributors-data

The following `start-query-workload-insights-top-contributors-data` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
    }
```

For more information, see Evaluate network flows with workload insights.

start-query-workload-insights-top-contributors

The following `start-query-workload-insights-top-contributors` example starts the query which returns a queryId to retrieve the top contributors.

```
aws networkflowmonitor start-query-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --start-time 2024-12-09T19:00:00Z \
        --end-time 2024-12-09T19:15:00Z \
        --metric-name DATA_TRANSFERRED \
        --destination-category UNCLASSIFIED
```

Output:

```
{
        "queryId": "sample-dddd-eeee-ffff-44556677889"
```

```
    }
```

For more information, see Evaluate network flows with workload insights.

stop-query-monitor-top-contributors

The following `stop-query-monitor-top-contributors` example stops the query in the specified account.

```
aws networkflowmonitor stop-query-monitor-top-contributors \
        --monitor-name Demo \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see Evaluate network flows with workload insights.

stop-query-workload-insights-top-contributors-data

The following `stop-query-workload-insights-top-contributors-data` stops the query in the specified account.

```
aws networkflowmonitor stop-query-workload-insights-top-contributors-data \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see Evaluate network flows with workload insights.

stop-query-workload-insights-top-contributors

The following `stop-query-workload-insights-top-contributors` example stops the query in the specified account.

```
aws networkflowmonitor stop-query-workload-insights-top-contributors \
        --scope-id sample-aaaa-bbbb-cccc-11112222333 \
        --query-id sample-dddd-eeee-ffff-44556677889
```

This command produces no output.

For more information, see Evaluate network flows with workload insights.

Tag a monitor

The following `tag-resource` adds a tag to the monitor in the specified account.

```
aws networkflowmonitor tag-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo \
        --tags Key=stack,Value=Production
```

This command produces no output.

For more information, see Tagging your Amazon CloudWatch resources.

Remove a tag from a monitor

The following `untag-resource` example removes a tag to the monitor in the specified account.

```
aws networkflowmonitor untag-resource \
        --resource-arn arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo \
        --tag-keys stack
```

This command produces no output.

For more information, see Tagging your Amazon CloudWatch resources.

Update an existing monitor

The following `update-monitor` example updates the monitor named ``Demo`` in the specified account.

```
aws networkflowmonitor update-monitor \
        --monitor-name Demo \
        --local-resources-to-add type="AWS::EC2::VPC",identifier="arn:aws:ec2:us-
east-1:111122223333:vpc/vpc-11223344556677889"
```

Output:

```
{
        "monitorArn": "arn:aws:networkflowmonitor:us-east-1:111122223333:monitor/Demo",
```

```
        "monitorName": "Demo",
        "monitorStatus": "ACTIVE",
        "tags": {
            "Value": "Production",
            "Key": "stack"
        }
    }
```

For more information, see Components and features of Network Flow Monitor.

Work with EKS

Using Network Flow Monitor, you can collect performance metrics for workloads that use Amazon Elastic Kubernetes Service (Amazon EKS). This chapter shows you how to install the agent step-by-step and describes the different EKS scenarios you can monitor. You'll also find detailed descriptions of the metadata that Network Flow Monitor provides for Amazon EKS in the console to help you understand network performance.

To gain the benefits of Network Flow Monitor performance monitoring, you must first install the AWS Network Flow Monitor Agent add-on for Amazon EKS. For more information, see Install the EKS AWS Network Flow Monitor Agent add-on.

If you want to monitor a single EKS cluster with enhanced visibility of workload traffic and performance insights within the cluster and with external destinations, see Amazon EKS Network Observability.

Contents

- Install the EKS AWS Network Flow Monitor Agent add-on
- Additional network path metadata included for Amazon EKS

Install the EKS AWS Network Flow Monitor Agent add-on

Follow the steps in this section to install the AWS Network Flow Monitor Agent add-on for Amazon Elastic Kubernetes Service (Amazon EKS), to send Network Flow Monitor metrics to the Network Flow Monitor backend from Kubernetes clusters. After you complete the steps, AWS Network Flow Monitor Agent pods will be running on all of your Kubernetes cluster nodes.

If you use self-managed Kubernetes clusters, the installation steps to follow are in the previous section: Install agents for self-managed Kubernetes instances.

You can install the add-on by using the console or by using API commands with the AWS Command Line Interface.

Contents

- Prerequisites for installing the add-on
- Install the AWS Network Flow Monitor Agent add-on by using the console
- Install the AWS Network Flow Monitor Agent add-on by using the AWS Command Line Interface
- Configure add-on for third party monitoring tools

Prerequisites for installing the add-on

Regardless of whether you use the console or the CLI to install the AWS Network Flow Monitor Agent add-on, there are several requirements for installing the add-on with Kubernetes.

Make sure that your version of Kubernetes is supported

Network Flow Monitor agent installation requires Kubernetes Version 1.25, or a more recent version.

Amazon EKS Pod Identity Agent add-on installation

You can install the Amazon EKS Pod Identity Agent add-on in the console or by using the CLI.

Amazon EKS Pod Identity associations provide the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Amazon EKS Pod Identity provides credentials to your workloads with an additional Amazon EKS Auth API and an agent pod that runs on each node.

To learn more and see the steps for installing the Amazon EKS Pod Identity add-on, see Set up the Amazon EKS Pod Identity Agent in the Amazon EKS User Guide.

Install the AWS Network Flow Monitor Agent add-on by using the console

Follow the steps in this section to install and configure the AWS Network Flow Monitor Agent add-on in the Amazon EKS console.

If you have already installed the add-on and have issues upgrading to a new version, see Troubleshoot issues in EKS agents installation.

Before you begin, make sure that you have installed the Amazon EKS Pod Identity Agent add-on. For more information, see the previous section.

To install the add-on using the console

1. In the AWS Management Console, navigate to the Amazon EKS console.
2. On the page for installing add-ons, in the list of add-ons, choose **AWS Network Flow Monitor Agent**.
3. Configure the add-on settings.
    1. For **Add-on access**, choose **EKS Pod Identity**.
    2. For the IAM role to use with the add-on, use a role that has the following AWS managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. This policy gives permission for an agent to send telemetry reports to the Network Flow Monitor endpoint. If you don't already have a role with the policy attached, create a role by choosing **Create recommended role**, and following the steps in the IAM console.
    3. Choose **Next**.
4. On the **Review and add** page, make sure that the add-on configuration looks correct, and then choose **Create**.

Install the AWS Network Flow Monitor Agent add-on by using the AWS Command Line Interface

Follow the steps in this section to install the AWS Network Flow Monitor Agent add-on for Amazon EKS by using the AWS Command Line Interface.

1. Install the EKS Pod Identity Agent add-on

Before you begin, make sure that you have installed the Amazon EKS Pod Identity Agent add-on. For more information, see the earlier section.

2. Create the required IAM role

The AWS Network Flow Monitor Agent add-on must have permission to send metrics to the Network Flow Monitor backend. You must attach a role with the required permissions when you create the add-on. Create a role that has the following AWS managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. You need the ARN of this IAM role to install the add-on.

3. Install the AWS Network Flow Monitor Agent add-on

To install the AWS Network Flow Monitor Agent add-on for your cluster, run the following command:

```
aws eks create-addon --cluster-name CLUSTER NAME --addon-name aws-network-flow-monitoring-agent --region AWS REGION --pod-identity-associations serviceAccount=aws-network-flow-monitor-agent-service-account,roleArn=IAM ROLE ARN
```

The result should be similar to the following:

```
{
    "addon": {
        "addonName": "aws-network-flow-monitoring-agent",
        "clusterName": "ExampleClusterName",
        "status": "CREATING",
        "addonVersion": "v1.0.0-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:us-west-2:000000000000:addon/ExampleClusterName/
aws-network-flow-monitoring-agent/eec11111-bbbb-EXAMPLE",
        "createdAt": "2024-10-25T16:38:07.213000+00:00",
        "modifiedAt": "2024-10-25T16:38:07.240000+00:00",
        "tags": {},
         "podIdentityAssociations": [
            "arn:aws:eks:us-west-2:000000000000:podidentityassociation/
ExampleClusterName/a-3EXAMPLE5555555"
         ]
    }
  }
```

4. Make sure that the add-on is active

Review the installed AWS Network Flow Monitor Agent add-on to make sure that it's active for your cluster. Run the following command to verify that the status is ACTIVE:

```
aws eks describe-addon --cluster-name CLUSTER NAME --addon-name aws-network-flow-monitoring-agent --region AWS REGION
```

The result should be similar to the following:

```
{
    "addon": {
        "addonName": "aws-network-flow-monitoring-agent",
        "clusterName": "ExampleClusterName",
        "status": "ACTIVE",
        "addonVersion": "v1.0.0-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:us-west-2:000000000000:addon/ExampleClusterName/
aws-network-flow-monitoring-agent/eec11111-bbbb-EXAMPLE",
        "createdAt": "2024-10-25T16:38:07.213000+00:00",
        "modifiedAt": "2024-10-25T16:38:07.240000+00:00",
        "tags": {},
         "podIdentityAssociations": [
            "arn:aws:eks:us-west-2:000000000000:podidentityassociation/
ExampleClusterName/a-3EXAMPLE5555555"
         ]
    }
  }
```

Configure add-on for third party monitoring tools

You can configure the Network Flow Monitor add-on to expose an OpenMetrics server during installation. This enables integration with third-party monitoring tools such as Prometheus, allowing you to collect and analyze network flow metrics alongside your existing monitoring infrastructure. Learn more about about OpenMetrics. This feature is available from add-on version v1.1.0.

To enable the OpenMetrics server, add OPEN_METRICS, OPEN_METRICS_ADDRESS, and OPEN_METRICS_PORT to the configuration values of the EKS Network Flow Monitor add-on. This guide will explain how to do this using both CLI and Console. See Amazon EKS add-ons advanced configuration for additional details about adding configuration values.

CLI Configuration

When using AWS Command Line Interface, you can provide the configuration values as a parameter:

```
aws eks create-addon \
  --cluster-name my-cluster-name \
```

```
  --addon-name aws-network-flow-monitoring-agent \
  --addon-version v1.1.0-eksbuild.1 \
  --configuration-values '{"env":
{"OPEN_METRICS":"on","OPEN_METRICS_ADDRESS":"0.0.0.0","OPEN_METRICS_PORT":9109}}'
```

Console Configuration

When using the Amazon EKS Console, these values can be added under Optional configuration settings, as part of the Configuration values.

Sample JSON:

```
{
    "env": {
        "OPEN_METRICS": "on",
        "OPEN_METRICS_ADDRESS": "0.0.0.0",
        "OPEN_METRICS_PORT": 9109
    }
}
```

Sample YAML:

```
env:
  OPEN_METRICS: "on"
  OPEN_METRICS_ADDRESS: "0.0.0.0"
  OPEN_METRICS_PORT: 9109
```

EKS Network Flow Monitor add-on OpenMetric Parameters

- **OPEN_METRICS:**
  - Enable or disable open metrics. Disabled if not supplied
  - Type: String
  - Values: ["on", "off"]
- **OPEN_METRICS_ADDRESS:**
  - Listening IP address for open metrics endpoint. Defaults to 127.0.0.1 if not supplied
  - Type: String
- **OPEN_METRICS_PORT:**
  - Listening port for open metrics endpoint. Defaults to 80 if not supplied

- Type: Integer
- Range: [0..65535]

Important: When setting OPEN_METRICS_ADDRESS to 0.0.0.0, the metrics endpoint will be accessible from any network interface. Consider using 127.0.0.1 for localhost-only access or implement appropriate network security controls to restrict access to authorized monitoring systems only.

Troubleshooting

If you encounter issues with the OpenMetrics server configuration, use the following information to diagnose and resolve common problems.

Metrics not displaying

Problem: The OpenMetrics server is configured, but metrics are not appearing in your monitoring tool.

Troubleshooting Steps:

1. Verify that the OpenMetrics server is enabled in your add-on configuration:
   - Check that OPEN_METRICS is set to "on" in your configuration values. See describe-addon.
   - Confirm that the add-on version is v1.1.0 or later in the *Configure selected add-ons settings*.
2. Test the metrics endpoint directly:
   - Access the metrics at http://*pod-ip:port*/metrics (replace pod-ip with the actual pod IP address and port with your configured port).
   - If you can't access the endpoint, verify your network configuration and security group settings.
3. Validate your monitoring tool configuration. Consult you monitoring tools user guide for details on how to do the following:
   - Make sure your monitoring tool (such as Prometheus) is configured to scrape the correct endpoint.
   - Check that the scraping interval and timeout settings are appropriate.
   - Verify that your monitoring tool has network access to the pod IP address.

Metrics missing from specific pods

Problem: Metrics are available from some pods but not others in your cluster.

Troubleshooting Steps:

The Network Flow Monitor add-on doesn't support pods that use hostNetwork: true. If your pod specification includes this setting, metrics won't be available from those pods.

Workaround: Remove the hostNetwork: true setting from your pod specification if possible. If you require host networking for your application, consider using alternative monitoring approaches for those specific pods.

Connection refused errors

Problem: You receive "connection refused" errors when trying to access the metrics endpoint.

Troubleshooting Steps:

1. Verify the OPEN_METRICS_ADDRESS configuration:
   - If set to 127.0.0.1, the endpoint is only accessible from within the pod.
   - If set to 0.0.0.0, the endpoint should be accessible from other pods in the cluster.
   - Make sure your monitoring tool can reach the configured address.
2. Check the OPEN_METRICS_PORT configuration:
   - Verify that the port number is not already in use by another service.
   - Make sure the port is within the valid range (1-65535).
   - Confirm that any security groups or network policies allow traffic on this port.

Verification steps

To confirm your OpenMetrics configuration is working correctly:

1. Check the add-on status:

```
aws eks describe-addon --cluster-name your-cluster-name --addon-name aws-network-flow-monitoring-agent
```

2. Verify pod status:

```
kubectl get pods app.kubernetes.io/name=aws-network-flow-monitoring-agent
```

3. Test the metrics endpoint from within the cluster:

```
kubectl exec add-on-pod-name -- curl localhost:9109/metrics
```

Replace 9109 with your configured port number, and the pod name with an AddOn pod name.

Additional network path metadata included for Amazon EKS

When Network Flow Monitor gathers performance metrics for network flows between Amazon EKS components, it includes additional metadata information about the network path, to help you better understand how the network paths for your workload are performing.

You can view detailed information about Amazon EKS network flow performance by creating a monitor for the network flows that you're interested in, and then viewing details on the **Historical explorer** tab.

With Network Flow Monitor, you can measure network performance between the following Amazon EKS components, to better understand how your workload is performing with your Amazon EKS configuration and determine where there are bottlenecks or impairments.

- Pod to pod on the same node
- Node to node on the same cluster
- Pod to pod on a different cluster
- Node to node on different clusters
- With and without Network Load Balancer

The following table lists the information that Network Flow Monitor returns for each network flow scenario.

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namesp	Pod name	Service	Namespace
Local pod connecting to cluster IP of another internal cluster service	Local	Local pod IP address	Remote pod IP address (through cluster IP address)	✓	✓	✓	✓ [1]	✓	✓
Local pod in a node network namespa connecting to cluster IP of another internal cluster service	Local	Local node IP address	Remote pod IP address (through cluster IP address)	✓ [2]	✓ [2]	✓ [2]	✓ [1]	✓	✓
Local pod connecting to individua	Local	Local pod IP address	Remote pod IP address	✓	✓	✓	✓	✓	✓

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namespa	Pod name	Service	Namespace
l pod IP address of another pod (headless service)									
Local pod connecting to individual pod IP address of another pod in node network namespace (headless service)	Local	Local pod IP address	Remote node IP address	✓	✓	✓	✓	✓	✓

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namespace	Pod name	Service	Namespace
Local pod connecting to remote pod in another cluster	Local	Local pod IP address	Remote pod IP address (another cluster)	✓	✓	✓	✗	✗	✗
Local pod connecting to an external network address	Local	Local pod IP address	External IP address	✓	✓	✓	N/A	N/A	N/A
Local pod operating in a node network namespace connecting to an external network IP address	Local	Local node IP address	External IP address	✓ [2]	✓ [2]	✓ [2]	N/A	N/A	N/A

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namespace	Pod name	Service	Namespace
Remote pod connecting to local pod through cluster IP address	Remote	Local pod IP address (through cluster IP address)	Remote pod IP address	✓	✓	✓	✓	✓	✓
Remote pod in a node network namespace connecting to local pod	Remote	Local pod IP address (through cluster IP address)	Remote node IP address	✓	✓	✓	✓ [3]	✓ [3]	✓ [3]
Remote pod connecting to local pod (headless service)	Remote	Local pod IP address	Remote pod IP address	✓	✓	✓	✓	✓	✓

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namespace	Pod name	Service	Namespace
External pod connecting to a local pod	Remote	Local pod IP address	Remote pod IP address	✓	✓	✓	✗	✗	✗
External resource connecting through NodePort or a Load Balancer to a local pod	Remote	Local pod IP address	External IP address [4]	✓	✓	✓	N/A	N/A	N/A

Connection information				Metadata information					
				Local			Remote		
Scenario	Initiated by	Local	Remote	Pod name	Service	Namespace	Pod name	Service	Namespace
External resource connecting through NodePort or a Load Balancer to a local pod operating in a node network namespace	Remote	Local node IP address	External IP address [4]	✓	✓	✓	N/A	N/A	N/A

Be aware of the following additional information corresponding to the items marked with footnotes in the preceding table.

1. Pod name is not visible in this scenario for pods with other owners, such as a Kubernetes service managed by the EKS control plane.
2. Local pod name, service, and namespace are not resolved if other pods are present in node network namespace.
3. Remote pod name, service, and namespace are not resolved if other pods are present in node network namespace.

4. If service is using NodePort or LoadBalancer in instance mode, and `ExternalTrafficPolicy` is set to `Cluster`, then this IP address will be reported as the IP address of the node that receives the NodePort connection.

Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances

To provide performance metrics for network flows in your AWS workloads, Network Flow Monitor relies on *agents* that you install, which send the metrics to Network Flow Monitor. You install Network Flow Monitor agents on your instances, and then set the correct permissions for the agents so that they can send metrics to the Network Flow Monitor backend.

An agent is a lightweight software application that you install on your resources, such as your VPC EC2 instances. Agents send performance metrics to the Network Flow Monitor backend on an ongoing basis. Then, you can view the metrics on the **Workload insights** page in the Network Flow Monitor console. You can also track detailed metrics for a specific network flow, or set of flows, by creating a monitor.

The steps that you follow to deploy agents in your instances depend on the type of instance: Amazon EKS Kubernetes instances, VPC EC2 instances, or self-managed (non-EKS) Kubernetes instances.

- For information about working with Amazon EKS, including installing agents on EKS, see Work with EKS.
- For information about installing agents on VPC EC2 instances and self-managed Kubernetes instances, see the sections in this chapter.

You can establish a private connection between your VPC and Network Flow Monitor agents by using AWS PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.

Contents

- Linux versions supported for Network Flow Monitor agents
- Install and manage agents for EC2 instances
- Install agents for self-managed Kubernetes instances
- Using AI agents to install and troubleshoot Network Flow Monitor agents

Linux versions supported for Network Flow Monitor agents

The instances that you install agents on must be running supported versions and distributions of Linux. Network Flow Monitor supports agents to run only on Linux, and the Linux kernel version must be 5.8 or later. The following Linux distributions are supported. Note that agents are tested to run on the latest versions of these distributions.

- Amazon Linux
- Ubuntu
- Red Hat
- Suse Linux
- Debian distributions for both x86 and aarch64

Install and manage agents for EC2 instances

Follow the steps in this section to install Network Flow Monitor agents for workloads on Amazon EC2 instances. Installing agents requires two steps: first, configure permissions to allow the agents to send metrics to the Network Flow Monitor backend, and then install the agents using one of the available methods.

Regardless of the method that you use to install agents on EC2 instances, you must configure permissions for the agents to enable them to send performance metrics to the Network Flow Monitor backend.

Step 1: Configure permissions

Before installing agents, set up the required IAM permissions. See Configure permissions for agents.

Step 2: Install agents

Choose one of the following methods to install the Network Flow Monitor agent on your EC2 instances:

- Install agents via CloudShell script – The fastest way to install agents. Select your instances in the Network Flow Monitor console and run an automated script in .
- Install agents by using CDK – Use AWS Cloud Development Kit (AWS CDK) to install agents as part of your infrastructure-as-code deployment.

- Install agents using SSM – Use AWS Systems Manager Distributor to install and manage agents from the console.
- Download and install the agent from the command line – Download prebuilt RPM or DEB packages and install them directly on your instances.
- Install agents using AWS CLI with SSM commands – Use the AWS Command Line Interface to send SSM commands to install and activate agents.

Contents

- Configure permissions for agents
- Install agents via CloudShell script
- Install agents on EC2 instances with SSM
- Download prebuilt packages of the Network Flow Monitor agent by using the command line
- Install agents using AWS CLI with SSM commands
- Install agents for EC2 instances by using CDK

Configure permissions for agents

To enable agents to send metrics to the Network Flow Monitor ingestion backend, the EC2 instances that the agents run in must use a role that has a policy attached with the correct permissions. To provide the required permissions, use a role that has the following AWS managed policy attached: CloudWatchNetworkFlowMonitorAgentPublishPolicy. Attach this policy to the IAM roles of the EC2 instances where you plan to install Network Flow Monitor agents.

We recommend that you add the permissions before you install agents on the EC2 instances. You can choose to wait until after you install agents, but the agents won't be able to send metrics to the service until the permissions are in place.

To add permissions for Network Flow Monitor agents

1. In the AWS Management Console, in the Amazon EC2 console, locate the EC2 instances that you plan to install Network Flow Monitor agents on.
2. Attach the CloudWatchNetworkFlowMonitorAgentPublishPolicy to the IAM role for each instance.

   If an instance doesn't have an IAM role attached, choose a role by doing the following:

1. Under **Actions**, choose **Security**.
2. Choose **Modify IAM role**, or create a new role by choosing **Create new IAM role**.
3. Choose a role for the instance, and attach the CloudWatchNetworkFlowMonitorAgentPublishPolicy policy.

Install agents via CloudShell script

The Network Flow Monitor console provides a one-click installation option that uses to automatically install and configure the Network Flow Monitor agent on your EC2 instances. This is the fastest way to get started with Network Flow Monitor, because the script handles the entire setup process for you—including configuring permissions, installing the agent, and activating it.

When you use this feature, you simply select the EC2 instances you want to monitor, and the console opens with a script that completes the full installation automatically. You don't need to manually configure IAM roles, run SSM commands, or download packages.

To install agents using the CloudShell script

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Install agents** page, in the **Install via script** panel, select one or more EC2 instances from the instance list.

   Only instances in a **running** state can be selected. Instances in other states (such as stopped or terminated) are shown but cannot be selected.
4. Choose **Install via CloudShell**.
5. opens and runs the installation script automatically. The script output shows the progress as it configures your instances.

After the script completes successfully, the agents begin collecting and sending performance metrics to the Network Flow Monitor backend. You should begin seeing data on the **Workload insights** page within approximately 20 minutes.

> **Important**
>
> The installation script requires that the AWS Systems Manager agent is installed and running on your EC2 instances. If your instances are in a private subnet, you must also have VPC endpoints configured for Systems Manager. For more information, see Working with Systems Manager Agent and Create VPC endpoints in the AWS Systems Manager User Guide.

Install agents on EC2 instances with SSM

Network Flow Monitor agents provide performance metrics about network flows. Follow the steps in this section to install and work with Network Flow Monitor agents on EC2 instances, by using AWS Systems Manager. If you use Kubernetes, skip to the next sections for information about installing agents with Amazon EKS clusters or self-managed Kubernetes clusters.

Network Flow Monitor provides a Distributor package for you in Systems Manager to use to install or uninstall agents. In addition, Network Flow Monitor provides a document to activate or deactivate agents, by using the Document Type command. Use standard Systems Manager procedures to use the package and the document, or follow the steps provided here for detailed guidance.

For more information in general about using Systems Manager, see the following documentation:

- AWS Systems Manager Run Command
- AWS Systems Manager Distributor

Complete the steps in the following sections to configure permissions, install, and work with Network Flow Monitor agents.

Contents

- Install or uninstall agents
- Activate or deactivate agents

Install or uninstall agents by using Systems Manager

Network Flow Monitor provides a distributor package in AWS Systems Manager for you to install Network Flow Monitor agents: **AmazonCloudWatchNetworkFlowMonitorAgent**. To access and run the package to install agents, follow the steps provided here.

To install agents in EC2 instances

1. In the AWS Management Console, in AWS Systems Manager, under **Node Tools**, choose **Distributor**.
2. Under **Owned by Amazon**, locate the Network Flow Monitor package, **AmazonCloudWatchNetworkFlowMonitorAgent**, and select it.
3. In the **Run command** flow, choose **Install one time** or **Install on schedule**.
4. In the **Target selection** section, choose how you want to select your EC2 instances to install agents on. You can select instances based on tags, choose instances manually, or base the choice on resource groups.
5. In the **Commmand parameters** section, under **Action**, choose **Install**.
6. Scroll down, if necessary, and then choose **Run** to start the installation.

If the installation is successful and the instances have permissions to access Network Flow Monitor endpoints, the agent will start collecting metrics and send reports to the Network Flow Monitor backend.

Agents that are active (sending metrics data) incur billing costs. For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the Amazon CloudWatch pricing page. If you don't need metrics data temporarily, you can deactivate an agent. For more information, see Activate or deactivate agents. If you no longer need Network Flow Monitor agents, you can uninstall them from the EC2 instances.

To uninstall agents from EC2 instances

1. In the AWS Management Console, in AWS Systems Manager, under **Node Tools**, choose **Distributor**.
2. Under **Owned by Amazon**, locate the Network Flow Monitor package, **AmazonCloudWatchNetworkFlowMonitorAgent**, and select it.
3. In the **Commmand parameters** section, under **Action**, choose **Uninstall**.
4. Select the EC2 instances to uninstall agents from.

5. Scroll down, if necessary, and then choose **Run** to start the installation.

Activate or deactivate agents by using Systems Manager

After you install a Network Flow Monitor agent with SSM, you must activate it to receive network flow metrics from the instance where it's installed. Agents that are active (sending metrics data) incur billing costs. For more information about Network Flow Monitor and Amazon CloudWatch pricing, see Network Monitoring on the Amazon CloudWatch pricing page. If you don't need metrics data temporarily, you can deactivate an agent to prevent ongoing billing for the agent.

Network Flow Monitor provides a document in AWS Systems Manager that you can use activate or deactivate agents that you've installed on your EC2 instances. By running this document to manage the agents, you can activate them to begin receiving performance metrics. Or, you can deactivate them to temporarily stop metrics from being sent,without uninstalling the agents.

The document in SSM that you can use to activate or deactivate agents is called **AmazonCloudWatch-NetworkFlowMonitorManageAgent**. To access and run the document, follow the steps in the procedure.

To activate or deactivate Network Flow Monitor agents

1. In the AWS Management Console, in AWS Systems Manager, under **Change Management Tools**, choose **Documents**.
2. Under **Owned by Amazon**, locate the Network Flow Monitor document, **AmazonCloudWatch-NetworkFlowMonitorManageAgent**, and select the document.
3. In the **Target selection** section, choose how you want to select your EC2 instances to install agents on. You can select instances based on tags, choose instances manually, or base the choice on resource groups.
4. In the **Command parameters** section, under **Action**, choose **Activate** or **Deactivate**, depending on the action that you want to take for the agents.
5. Scroll down, if necessary, and then choose **Run** to start the installation.

Download prebuilt packages of the Network Flow Monitor agent by using the command line

You can use the command line to install the Network Flow Monitor agent as a package in Amazon Linux 2023, or download and install prebuilt packages of the Network Flow Monitor agent.

Before or after you download a prebuilt package, you can optionally verify the package signature. For more information, see Verify the signature of the Network Flow Monitor agent package.

Choose from the following instructions, depending on the Linux operating system that you're using and the type of installation that you want.

Amazon Linux AMIs

The Network Flow Monitor agent is available as a package in Amazon Linux 2023. If you're using this operating system, you can install the package by entering the following command:

```
sudo yum install network-flow-monitor-agent
```

You must also make sure that the IAM role attached to the instance has the CloudWatchNetworkFlowMonitorAgentPublishPolicy policy attached. For more information, see Configure permissions for agents.

Amazon Linux 2023

Install the package for your architecture by using one of the following commands:

- **x86_64**: `sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm`
- **ARM64 (Graviton)**: `sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm`

Verify that Network Flow Monitor agent is successfully installed by running the following command and verifying that the response shows that the agent is enabled and active:

```
service network-flow-monitor status
network-flow-monitor.service - Network Flow Monitor Agent
     Loaded: loaded (/usr/lib/systemd/system/network-flow-monitor.service; enabled;
 preset: enabled)
     Active: active (running) since Wed 2025-04-23 19:17:16 UTC; 1min 9s ago
```

DEB-based distributions (Debian, Ubuntu)

Install the package for your architecture by using one of the following commands:

- **x86_64**: `wget https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb`

- **ARM64 (Graviton)**: `wget https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb`

Install the package by using the following command: `$ sudo apt-get install ./network-flow-monitor-agent.deb`

Verify that Network Flow Monitor agent is successfully installed by running the following command and verifying that the response shows that the agent is enabled and active:

```
service network-flow-monitor status
network-flow-monitor.service - Network Flow Monitor Agent
     Loaded: loaded (/usr/lib/systemd/system/network-flow-monitor.service; enabled;
 preset: enabled)
     Active: active (running) since Wed 2025-04-23 19:17:16 UTC; 1min 9s ago
```

Verify the signature of the Network Flow Monitor agent package

The Network Flow Monitor agent rpm and deb installer packages for Linux instances are cryptographically signed. You can use a public key to verify that the agent package is original and unmodified. If the files are damaged or have been altered, the verification fails. You can verify the signature of the installer package using either RPM or GPG. The following information is for Network Flow Monitor agent versions 0.1.3 or later.

To find the correct signature file for each architecture and operating system, use the following table.

Architecture	Platform	Download link	Signature file link
x86-64	Amazon Linux 2023	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.rpm.sig
ARM64	Amazon Linux 2023	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.rpm.sig

Architecture	Platform	Download link	Signature file link
x86-64	Debian/Ubuntu	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb	https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-monitor-agent.deb.sig
ARM64	Debian/Ubuntu	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb	https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-flow-monitor-agent.deb.sig

Follow the steps here to verify the signature of the Network Flow Monitor agent.

To verify the signature of the Network Flow Monitor agent for Amazon S3 package

1. Install GnuPG so that you can run the gpg command. GnuPG is required to verify the authenticity and integrity of a downloaded Network Flow Monitor agent for an Amazon S3 package. GnuPG is installed by default on Amazon Linux Amazon Machine Images (AMIs).
2. Copy the following public key and save it to a file named `nfm-agent.gpg`.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGf0b5IBEAC6YQc0aYrTbcHNWWMbLuqsqfspzWrtCvoU0yQ62ld7nvCGBha9
lu4lbhtiwoDawC3h6Xsxc3Pmm6kbMQfZdbo4Gda4ahf6zDOVI5zVHs3Yu2VXC2AU
5BpKQJmYddTb7dMI3GBgEodJY05NHQhq1Qd2ptdh03rsX+96Fvi4A6t+jsGzMLJU
I+hGEKGif69pJVyptJSibK5bWCDXh3eS/+vB/CbXumAKi0sq4rXv/VPiIhn6bsCI
A2lmzFd3vMJQUM/T7m7skrqetZ4mWHr1LPDFPK/H/81s8TJawx7MACsK6kIRUxu+
oicW8Icmg9S+BpIgONT2+Io5P1tYO5a9AyVF7X7gU0VgHUA1RoLnjHQHXbCmnFtW
cYEuwhUuENMl+tLQCZ+fk0kKjOlIKqeS9AVwhks92oETh8wpTwTE+DTBvUBP9aHo
S39RTiJCnUmA6ZCehepgpwW9AYCc1lHv/xcahD418E0UHV22qIw943EwAkzMDA4Q
damdRm0Nud0OmilCjo9oogEB+NUoy//5XgQMH1hhfsHquVLU/tneYexXYMfo/Iu5
TKyWL2KdkjKKP/dMR4lMAXYi0RjTJJ5tg5w/VrHhrHePFfKdYsgN6pihWwj2Px/M
ids3W1Ce50LOEBc2MOKXYXGd9OZWyR8l15ZGkySvLqVlRGwDwKGMC/nS2wARAQAB
tEJOZXR3b3JrIEZsb3cgTW9uaXRvciBBZ2VudCA8bmV0d29yay1mbG93LW1vbml0
b3ItYWdlbnRAYW1hem9uLmNvbT6JAlcEEwEIAEEWIQR2c2ypl63T6dJ3JqjvvaTM
vJX60QUCZ/RvkgIbAwUJBaOagAULCQgHAgIiAgYVCgkICwIEFgIDAQIeBwIXgAAK
CRDvvaTMvJX60euSD/9cIu2BDL4+MFFHhyHmG3/se8+3ibW0g8SyP3hsnq7qN+bm
ZzLAhll7DVoveNmEHI1VC7Qjwb30exgLcyK2Ld6uN6lwjjK0qiGGz943t230pJ3z
u7V2fVtAN+vgDVmD7agE6iqrRCWu3WfcgzFlEkE/7nkhtbWzlaK+NkdEBzNZ+W7/
```

```
FmLClzIbMjIBW2M8LdeZdQX0SWljy18x7NGNukWeNTJxmkDsjAeKl+zkXYk9h7ay
n3AVl1KrLZ5P9vQ5XsV5e4T6qfQ3XNY1lm54cpa+eD7NyYcTGRDK+vIxO4xD8i2M
yl1iNf2+84Tt6/SAgR/P9SJ5tbKD0iU9n4g1eBJVGmHDuXTtDR4H/Ur7xRSxtuMl
yZP/sLWm8p7+Ic7aQJ5OVw36MC7Oa7/K/zQEnLFFPmgBwGGiNiw5cUSyCBHNvmtv
FK0Q2XMXtBEBU9f44FMyzNJqVdPywg8Y6xE4wc/68uy7G6PyqoxDSP2ye/p+i7oi
OoA+OgifchZfDVhe5Ie0zKR0/nMEKTBV0ecjglb/WhVezEJgUFsQcjfOXNUBesJW
a9kDGcs3jIAchzxhzp/ViUBmTg6SoGKh3t+3uG/RK2ougRObJMW3G+DI7xWyY+3f
7YsLm0eDd3dAZG3PdltMGp0hKTdslvpws9qoY8kyR0Fau4l222JvYP27BK44qg==
=INr5
-----END PGP PUBLIC KEY BLOCK-----
```

3. Import the public key into your keyring and note the returned value.

```
PS>  rpm --import nfm-agent.gpg
gpg: key 3B789C72: public key "Network Flow Monitor Agent" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
```

Make a note of the key value because you need it in the next step. In this example, the key value is 3B789C72.

4. Verify the fingerprint by running the following command. Be sure to replace *key-value* with the value from the preceding step. You recommend that you use GPG to verify the fingerprint even if you use RPM to verify the installer package.

```
PS>  gpg --fingerprint key-value
pub   rsa4096 2025-04-08 [SC] [expires: 2028-04-07]
      7673 6CA9 97AD D3E9 D277  26A8 EFBD A4CC BC95 FAD1
uid   Network Flow Monitor Agent <network-flow-monitor-agent@amazon.com>
```

The fingerprint string should be equal to the following:

```
7673 6CA9 97AD D3E9 D277 26A8 EFBD A4CC BC95 FAD1
```

If the fingerprint string doesn't match, don't install the agent. Contact Amazon Web Services.

After you have verified the fingerprint, you can use it to verify the signature of the Network Flow Monitor agent package.

5. Download the package signature file, if you haven't already done so, based on your instance's architecture and operating system.

6. Verify the installer package signature. Be sure to replace the `signature-filename` and `agent-download-filename` with the values that you specified when you downloaded the signature file and agent, as shown in the table earlier in this topic.

```
PS> gpg --verify sig-filename agent-download-filename
gpg: Signature made Tue Apr  8 00:40:02 2025 UTC
gpg:                using RSA key 77777777EXAMPLEKEY
gpg:                issuer "network-flow-monitor-agent@amazon.com"
gpg: Good signature from "Network Flow Monitor Agent <network-flow-monitor-
agent@amazon.com>" [unknown]
gpg: WARNING: Using untrusted key!
```

   If the output includes the phrase BAD `signature`, check to make sure that you performed the procedure correctly. If you continue to get this response, contact AWS Support and avoid using the downloaded file.

   Note the warning about trust. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Next, follow the steps here to verify the RPM package.

To verify the signature of the RPM package

1. Copy the following public key and save it to a file named `nfm-agent.gpg`.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGf0b5IBEAC6YQc0aYrTbcHNWWMbLuqsqfspzWrtCvoU0yQ62ld7nvCGBha9
lu4lbhtiwoDawC3h6Xsxc3Pmm6kbMQfZdbo4Gda4ahf6zDOVI5zVHs3Yu2VXC2AU
5BpKQJmYddTb7dMI3GBgEodJY05NHQhq1Qd2ptdh03rsX+96Fvi4A6t+jsGzMLJU
I+hGEKGif69pJVyptJSibK5bWCDXh3eS/+vB/CbXumAKi0sq4rXv/VPiIhn6bsCI
A2lmzFd3vMJQUM/T7m7skrqetZ4mWHr1LPDFPK/H/81s8TJawx7MACsK6kIRUxu+
oicW8Icmg9S+BpIgONT2+Io5P1tYO5a9AyVF7X7gU0VgHUA1RoLnjHQHXbCmnFtW
cYEuwhUuENMl+tLQCZ+fk0kKjOlIKqeS9AVwhks92oETh8wpTwTE+DTBvUBP9aHo
S39RTiJCnUmA6ZCehepgpwW9AYCc1lHv/xcahD418E0UHV22qIw943EwAkzMDA4Q
damdRm0Nud0OmilCjo9oogEB+NUoy//5XgQMH1hhfsHquVLU/tneYexXYMfo/Iu5
TKyWL2KdkjKKP/dMR4lMAXYi0RjTJJ5tg5w/VrHhrHePFfKdYsgN6pihWwj2Px/M
ids3W1Ce50LOEBc2MOKXYXGd9OZWyR8l15ZGkySvLqVlRGwDwKGMC/nS2wARAQAB
tEJOZXR3b3JrIEZsb3cgTW9uaXRvciBBZ2VudCA8bmV0d29yay1mbG93LW1vbml0
b3ItYWdlbnRAYW1hem9uLmNvbT6JAlcEEwEIAEEWIQR2c2ypl63T6dJ3JqjvvaTM
```

```
vJX60QUCZ/RvkgIbAwUJBaOagAULCQgHAgIiAgYVCgkICwIEFgIDAQIeBwIXgAAK
CRDvvaTMvJX60euSD/9cIu2BDL4+MFFHhyHmG3/se8+3ibW0g8SyP3hsnq7qN+bm
ZzLAhll7DVoveNmEHI1VC7Qjwb30exgLcyK2Ld6uN6lwjjK0qiGGz943t230pJ3z
u7V2fVtAN+vgDVmD7agE6iqrRCWu3WfcgzFlEkE/7nkhtbWzlaK+NkdEBzNZ+W7/
FmLClzIbMjIBW2M8LdeZdQX0SWljy18x7NGNukWeNTJxmkDsjAeKl+zkXYk9h7ay
n3AVl1KrLZ5P9vQ5XsV5e4T6qfQ3XNY1lm54cpa+eD7NyYcTGRDK+vIxO4xD8i2M
yl1iNf2+84Tt6/SAgR/P9SJ5tbKD0iU9n4g1eBJVGmHDuXTtDR4H/Ur7xRSxtuMl
yZP/sLWm8p7+Ic7aQJ5OVw36MC7Oa7/K/zQEnLFFPmgBwGGiNiw5cUSyCBHNvmtv
FK0Q2XMXtBEBU9f44FMyzNJqVdPywg8Y6xE4wc/68uy7G6PyqoxDSP2ye/p+i7oi
OoA+OgifchZfDVhe5Ie0zKR0/nMEKTBV0ecjglb/WhVezEJgUFsQcjfOXNUBesJW
a9kDGcs3jIAchzxhzp/ViUBmTg6SoGKh3t+3uG/RK2ougRObJMW3G+DI7xWyY+3f
7YsLm0eDd3dAZG3PdltMGp0hKTdslvpws9qoY8kyR0Fau4l222JvYP27BK44qg==
=INr5
-----END PGP PUBLIC KEY BLOCK-----
```

2. Import the public key into your keyring.

```
PS>  rpm --import nfm-agent.gpg
```

3. Verify the installer package signature. Be sure to replace the `agent-download-filename` with the value that you specified when you downloaded the agent, as shown in the table earlier in this topic.

```
PS>  rpm --checksig agent-download-filename
```

For example, for the x86_64 architecture on Amazon Linux 2023, use the following command:

```
PS>  rpm --checksig network-flow-monitor-agent.rpm
```

This command returns output similar to the following.

```
network-flow-monitor-agent.rpm: digests signatures OK
```

If the output contains the phrase `NOT OK (MISSING KEYS: (MD5) key-id)`, check to make sure that you performed the procedure correctly. If you continue to get this response, contact AWS Support and don't install the agent.

Install agents using AWS CLI with SSM commands

You can use the AWS Command Line Interface to send AWS Systems Manager commands to install and activate Network Flow Monitor agents on your EC2 instances. This method is useful for scripted or automated deployments.

Step 1: Install the agent

Use the following AWS CLI command to install the Network Flow Monitor agent on one or more EC2 instances using the `AWS-ConfigureAWSPackage` SSM document:

```
aws ssm send-command --document-name "AWS-ConfigureAWSPackage" \
  --parameters '{"action":["Install"],"installationType":["Uninstall and
 reinstall"],"name":["AmazonCloudWatchNetworkFlowMonitorAgent"],"version":
[""],"additionalArguments":["{}"]}' \
  --targets "Key=instanceids,Values=i-1234567890abcdef0"
```

Replace *i-1234567890abcdef0* with the instance ID (or IDs) where you want to install the agent. You can specify multiple instance IDs separated by commas.

Step 2: Activate the agent

After the installation completes, activate the agent so it begins sending performance metrics to the Network Flow Monitor backend. Activating the agent incurs billing costs. For more information about pricing, see the Amazon CloudWatch pricing page.

```
aws ssm send-command --document-name "AmazonCloudWatch-NetworkFlowMonitorManageAgent" \
  --parameters '{"Action":["Activate"]}' \
  --targets "Key=instanceids,Values=i-1234567890abcdef0"
```

After the agent is activated, it begins collecting and sending performance metrics. You should see data on the **Workload insights** page within approximately 20 minutes.

Install agents for EC2 instances by using CDK

You can use AWS Cloud Development Kit (AWS CDK) to install the Network Flow Monitor agent as part of your infrastructure-as-code deployment. Add the agent installation to your EC2 instance user data in your CDK stack.

The following example shows how to install the Network Flow Monitor agent on an Amazon Linux instance using CDK with TypeScript:

```
// Create an EC2 instance
const ec2Instance = new ec2.Instance(this, 'Instance', {
  vpc,
  instanceType: ec2.InstanceType.of(ec2.InstanceClass.BURSTABLE3,
 ec2.InstanceSize.MICRO),
  machineImage: ec2.MachineImage.latestAmazonLinux2023(),
  vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
});

// Install Network Flow Monitor Agent via RPM on Amazon Linux (x86_64)
ec2Instance.addUserData(
  "sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/x86_64/
network-flow-monitor-agent.rpm -y"
);

// Attach the required permissions policy to your EC2 instance role
ec2Instance.role.addManagedPolicy(

 iam.ManagedPolicy.fromAwsManagedPolicyName('CloudWatchNetworkFlowMonitorAgentPublishPolicy')
);
```

For ARM64 (Graviton) instances, replace the RPM URL with the ARM64 variant:

```
ec2Instance.addUserData(
  "sudo yum install https://networkflowmonitoragent.awsstatic.com/latest/arm64/network-
flow-monitor-agent.rpm -y"
);
```

For Debian/Ubuntu instances, use the DEB package instead:

```
ec2Instance.addUserData(
  "wget https://networkflowmonitoragent.awsstatic.com/latest/x86_64/network-flow-
monitor-agent.deb && sudo apt-get install ./network-flow-monitor-agent.deb -y"
);
```

Install agents for self-managed Kubernetes instances

Follow the steps in this section to install Network Flow Monitor agents for workloads on self-managed Kubernetes clusters. After you complete the steps, Network Flow Monitor agent pods will be running on all of your self-managed Kubernetes cluster nodes.

If you use Amazon Elastic Kubernetes Service (Amazon EKS), the installation steps to follow are in the following section: Install the EKS AWS Network Flow Monitor Agent add-on.

Contents

- Before you begin
- Download Helm charts and install agents
- Configure permissions for agents to deliver metrics

Before you begin

Before you start the installation process, follow the steps in this section to make sure that your environment is set up to successfully install agents on the right Kubernetes clusters.

Ensure that your version of Kubernetes is supported

Network Flow Monitor agent installation requires Kubernetes Version 1.25, or a more recent version.

Ensure that you have installed required tools

The scripts that you use for this installation process require that you install the following tools. If you don't have the tools installed already, see the provided links for more information.

- The AWS Command Line Interface (CLI). For more information, see Installing or updating to the latest version of the AWS Command Line Interface in the AWS Command Line Interface Reference Guide.
- The Helm package manager. For more information, see Installing Helm on the Helm website.
- The `kubectl` command line tool. For more information, see Install kubectl on the Kubernetes website.
- The `make` Linux command dependency. For more information, see the following blog post: Intro to make Linux Command: Installation and Usage. For example, do one of the following:
  - For Debian based distributions, such as Ubuntu, use the following command: `sudo apt-get install make`
  - For RPM-based distributions, such as CentOS, use the following command: `sudo yum install make`

Ensure that you have valid, correctly configured KubeConfig environment variables

Network Flow Monitor agent installation uses the Helm package manager tool, which uses the kubeconfig variable, `$HELM_KUBECONTEXT`, to determine the target Kubernetes clusters to work with. Also, be aware that when Helm runs installation scripts, by default, it references the standard `~/.kube/config` file. You can change the configuration environment variables, to use a different config file (by updating `$KUBECONFIG`) or to define the target cluster you want to work with (by updating `$HELM_KUBECONTEXT`).

Create a Network Flow Monitor Kubernetes namespace

The Network Flow Monitor agent's Kubernetes application installs its resources into a specific namespace. The namespace must exist for the installation to succeed. To ensure that the required namespace is in place, you can do one of the following:

- Create the default namespace, `amazon-network-flow-monitor`, before you begin.
- Create a different namespace, and then define it in the `$NAMESPACE` environment variable when you run the installation to make targets.

Download Helm charts and install agents

You can download the Network Flow Monitor agent Helm charts from the AWS public repository by using the following command. Make sure that you first authenticate with your GitHub account.

```
git clone https://github.com/aws/network-flow-monitor-agent.git
```

In the `./charts/amazon-network-flow-monitor-agent` directory, you can find the Network Flow Monitor agent Helm charts and Makefile that contain the installation make targets that you use for installing agents. You install agents for Network Flow Monitor by using the following Makefile target: `helm/install/customer`

You can customize the installation if you like, for example, by doing the following:

```
# Overwrite the kubeconfig files to use
KUBECONFIG=<MY_KUBECONFIG_ABS_PATH> make helm/install/customer

# Overwrite the Kubernetes namespace to use
NAMESPACE=<MY_K8S_NAMESPACE> make helm/install/customer
```

To verify that the Kubernetes application pods for the Network Flow Monitor agents have been created and deployed successfully, check to be sure that their state is `Running`. You can check

state of the agents by running the following command: `kubectl get pods -o wide -A | grep amazon-network-flow-monitor`

Configure permissions for agents to deliver metrics

After you install agents for Network Flow Monitor, you must enable the agents to send network metrics to the Network Flow Monitor ingestion APIs. Agents in Network Flow Monitor must have permission to access the Network Flow Monitor ingestion APIs so that they can deliver network flow metrics that they've collected for each instance. You grant this access by implementing IAM roles for service accounts (IRSA).

To enable agents to deliver network metrics to Network Flow Monitor, follow the steps in this section.

1. **Implement IAM roles for service accounts**

   IAM roles for service accounts provides the ability to manage credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Implementing IRSA is the recommended way to provide all permissions required by Network Flow Monitor agents to successfully access Network Flow Monitor ingestion APIs. For more information, see IAM roles for service accounts in the Amazon EKS User Guide.

   When you set up IRSA for Network Flow Monitor agents, use the following information:

   - **ServiceAccount:** When you define your IAM role trust policy, for `ServiceAccount`, specify `aws-network-flow-monitor-agent-service-account`.
   - **Namespace:** For the `namespace`, specify `amazon-network-flow-monitor`.
   - **Temporary credentials deployment:** When you configure permissions after you have deployed Network Flow Monitor agent pods, updating the `ServiceAccount` with your IAM role, Kubernetes does not deploy the IAM role credentials. To make sure that the Network Flow Monitor agents acquire the IAM role credentials that you've specified, you must rolling out a restart of `DaemonSet`. For example, use a command like the following:

     ```
     kubectl rollout restart daemonset -n amazon-network-flow-monitor aws-network-flow-monitor-agent
     ```

2. **Confirm that the Network Flow Monitor agent is successfully accessing the Network Flow Monitor ingestion APIs**

   You can check to make sure that your configuration for agents is working correctly by using the HTTP 200 logs for Network Flow Monitor agent pods. First, search for a Network Flow Monitor

agent pod, and then search through the log files to find successful HTTP 200 requests. For example, you can do the following:

a. Locate a Network Flow Monitor agent pod name. For example, you can use the following command:

```
RANDOM_AGENT_POD_NAME=$(kubectl get pods -o wide -A | grep amazon-network-flow-
monitor | grep Running | head -n 1 | tr -s ' ' | cut -d " " -f 2)
```

b. Grep all the HTTP logs for the pod name that you've located. If you've changed the NAMESPACE, make sure that you use the new one.

```
NAMESPACE=amazon-network-flow-monitor
kubectl logs $RANDOM_AGENT_POD_NAME -\-namespace ${NAMESPACE} | grep HTTP
```

If access has been granted successfully, you should see log entries similar to the following:

```
...
{"level":"INFO","message":"HTTP request
 complete","status":200,"target":"amzn_nefmon::reports::publisher_endpoint","timestamp":17370
{"level":"INFO","message":"HTTP request
 complete","status":200,"target":"amzn_nefmon::reports::publisher_endpoint","timestamp":17370
```

Note that the Network Flow Monitor agent publishes network flow reports every 30 seconds, by calling the Network Flow Monitor ingestion APIs.

Using AI agents to install and troubleshoot Network Flow Monitor agents

You can use Model Context Protocol (MCP)-compatible AI agents to install, configure, and troubleshoot Network Flow Monitor agents on your workloads. The `aws-network-monitoring` skill from the Agent Toolkit for AWS can help with the following:

- **Agent installation** – Install Network Flow Monitor agents using AWS Systems Manager Distributor or command-line install. The skill guides the assistant through attaching the `CloudWatchNetworkFlowMonitorAgentPublishPolicy` managed policy to the instance role, running the install document, and activating the agents.

- **Troubleshooting** – Diagnose common issues, including HTTP 403 errors from missing IAM permissions, missing performance metrics after installation (activation not run), and connectivity failures from private subnets that lack the required virtual private cloud (VPC) endpoints.

Prerequisites

Before you start, make sure you have the following:

- **An AI agent** – An AI coding agent that supports MCP, such as Kiro CLI. For a list of supported AI coding agents, see What is the Agent Toolkit for AWS? in the *AWS Agent Toolkit User Guide*.
- **AWS MCP server** – Configure the AWS MCP server in your agent so it can make AWS API calls. For more information, see Setting up the AWS MCP Server in the *AWS Agent Toolkit User Guide*.
- **AWS credentials** – An AWS account with IAM credentials set up on your local machine. Credentials are required for tools that execute AWS API calls and run scripts, but not for searching documentation or discovering skills.

Install the aws-network-monitoring skill

A skill is a curated package of instructions and reference materials that helps your AI agent complete Network Flow Monitor tasks. The `aws-network-monitoring` skill contains install procedures, IAM policy attachment, activation steps, and troubleshooting patterns. For more information, see Skills in the *AWS Agent Toolkit User Guide*.

Use the AWS CLI to install the skill for your agent. For more information about the install command, see Using the AWS CLI in the *AWS Agent Toolkit User Guide*.

```
aws agent-toolkit add-skill --skill-name aws-network-monitoring
```

After the skill is installed, your agent can load it on demand when you ask about installing, configuring, or troubleshooting Network Flow Monitor agents.

Example prompts

The following prompts show common Network Flow Monitor workflows you can run through an AI agent after installing the skill.

Install agents on EC2 instances

Provide an Amazon EC2 tag key and value, or one or more instance IDs, and the AWS Region. The agent resolves the target instances, runs the Systems Manager install document, attaches the required IAM policy to each instance role, and activates the agents.

```
Install Network Flow Monitor agents on my EC2 instances tagged Environment=Sandbox in
 us-east-1.
```

```
Install the Network Flow Monitor agent on my EC2 instance i-abc123 in us-east-1.
```

Verify that agents are publishing performance metrics

Ask the agent to check whether a running Network Flow Monitor agent is successfully publishing reports to the Network Flow Monitor ingestion endpoint.

```
Verify that the Network Flow Monitor agent on i-abc123 in us-east-1 is publishing
 performance metrics.
```

Troubleshoot agents that are not sending performance metrics

The agent checks the IAM policy attachment, activation status, and endpoint reachability.

```
My Network Flow Monitor agent on i-abc123 in us-east-1 is installed but not sending
 performance metrics. What's wrong?
```

Initialize Network Flow Monitor

Before you can view performance metrics for network flows, you must initialize Network Flow Monitor, which grants required permissions and creates an initial topology for your account or accounts. If you plan to monitor resources for multiple accounts, you must also configure AWS Organizations with Amazon CloudWatch. Then, you specify accounts for your Network Flow Monitor scope, so that Network Flow Monitor can create an initial topology for all the accounts that you'll be tracking performance metrics for.

In addition, you must install agents on your instances, to send performance metrics to the Network Flow Monitor ingestion server. For more information, see Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances.

The steps that you take to initialize Network Flow Monitor vary depending on whether you are measuring performance metrics for resources in a single account, or you want to monitor metrics from resources that are owned by multiple accounts in your organization.

- Single account monitoring initialization
- Multi-account monitoring initialization

Initialize Network Flow Monitor for single account monitoring

To initialize Network Flow Monitor to monitor network performance metrics, you must grant permissions and Network Flow Monitor must create the initial topology for your account. When you monitor resources in just one account, Network Flow Monitor sets your account as the scope for network monitoring and creates a topology for that scope.

Initializing Network Flow Monitor does the following:

- Grants permissions for Network Flow Monitor to use required service-linked roles with your account. Network Flow Monitor requires you to grant it specific permissions so that the feature can send metrics to Amazon CloudWatch on your behalf, as well as create topologies of network flows. For more information, see Service-linked roles for Network Flow Monitor.
- Sets your monitoring scope for Network Flow Monitor to the AWS account that you're signed in with. For more information, see **Scope** in Components and features of Network Flow Monitor.
- Creates an initial topology for your scope.

To initialize Network Flow Monitor by setting up the service-linked roles that provide the required permissions, setting the scope to your account, and creating a topology for network flow performance monitoring, follow these steps.

To initialize Network Flow Monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. In the **Getting started with Network Flow Monitor** section, under Step 1, choose **Start initialization**.
4. On the **Configure Network Flow Monitor** page, scroll down, and then choose **Initialize Network Flow Monitor**.

Completing the initialization can take 20-30 minutes.

After you initialize Network Flow Monitor for your account, before you can view network flow performance metrics, you must also install Network Flow Monitor agents for your resources

that send performance metrics to the Network Flow Monitor backed ingestion server. For more information, see Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances.

Initialize Network Flow Monitor for multi-account monitoring

If you want to monitor network flows in Network Flow Monitor for resources that are owned by different accounts, you must first configure Amazon CloudWatch with AWS Organizations. To use multiple accounts in Network Flow Monitor, you're required to turn on trusted access for CloudWatch, and it's a best practice to also register a delegated administrator.

In addition, if you plan to create monitors for network flows from the console, you must add a Network Flow Monitor policy to the role that is attached to your resources. The policy enables you to view resources from other accounts in the console, so that you can add the resources in multiple accounts to a monitor.

To monitor network flows for resources that are owned by different accounts, there are additional configuration steps to take. First, as the management account, you must configure CloudWatch with AWS Organizations to turn on trusted access, and, typically, you'll also register a delegated administrator account. Then, using the delegated administrator account, you can add more accounts in your organization, to set the scope for your network observability to include resources in those accounts. (You can also add multiple accounts with a management account. However, it's a best practice in Organizations to use the delegated administrator account when you work with resources in a service. We provide steps that follow that guidance in the instructions here for Network Flow Monitor.)

Note that if you don’t need to monitor network flows for instances from multiple accounts, you can use Network Flow Monitor with a single account. The scope for Network Flow Monitor is automatically set to the AWS account that you sign in with.

Use the guidance in the following sections to complete these steps.

⚠ Shared VPC with Amazon EKS clusters

If your Amazon EKS cluster runs in a member account that uses a shared subnet (a subnet owned by another account), you must create the monitor in the account that owns the subnet (typically the delegated admin or management account), not in the member account where the Amazon EKS cluster resides. A monitor created in the member account cannot match network flow data because the member account does not own the subnet

resources. To monitor flows for the Amazon EKS cluster, create the monitor in the subnet-owning account and specify the Amazon EKS cluster as the local resource.

Contents

- Overview of steps for using multiple accounts in Network Flow Monitor
- Configure AWS Organizations in CloudWatch
- Add multiple accounts to your scope
- Set up permissions for multi-account resource access (console only)

Overview of steps for using multiple accounts in Network Flow Monitor

To get started with Network Flow Monitor, any account that has not used Network Flow Monitor before must initialize Network Flow Monitor. When you initialize Network Flow Monitor for an account, Network Flow Monitor adds the required service-linked role permissions and creates a scope of the account or accounts to be included in network observability. To work with multiple accounts in Network Flow Monitor, there are additional steps, to integrate with AWS Organizations, and then add the accounts to work with.

In summary, you take the following steps:

1. Sign in to the AWS Management Console as the management account, and then do the following:
   - Complete the required steps for integrating with AWS Organizations in CloudWatch.
2. Sign in to the AWS Management Console as the delegated administrator account, and then do the following:
   - Initialize Network Flow Monitor, including adding accounts to include in your scope.
   - Add the required permissions for accessing resources that are in other accounts from the console.

If you're setting up Network Flow Monitor to work with multiple accounts and you're not familiar with AWS Organizations, review the following resources to learn about concepts such as the management account, trusted access, and the delegated administrator account, and to learn how to integrate Organizations with CloudWatch.

- Managing accounts in an organization with AWS Organizations in the AWS Organizations User Guide.
- Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

Follow the steps in the following sections for specific guidance in configuring Network Flow Monitor for multiple accounts.

Configure AWS Organizations in CloudWatch

To configure Network Flow Monitor with AWS Organizations, sign in to the management account, and turn on trusted access for CloudWatch. Then, register a delegated administrator account to use for initializing Network Flow Monitor and adding multiple accounts.

If you've already configured Organizations in CloudWatch to turn on trusted access for Organizations in CloudWatch and register a delegated administrator account, you don't need to configure anything more for Organizations that is specific to Network Flow Monitor. You can sign in with the delegated administrator account for CloudWatch, and then initialize Network Flow Monitor, including adding multiple accounts for your network observability scope.

If you haven't yet configured Organizations in CloudWatch, follow the steps here to turn on trusted access and register a delegated administrator account.

Turn on trusted access in CloudWatch

Before you can use Network Flow Monitor with more than one account in your organization, you must turn on trusted access for AWS Organizations in Amazon CloudWatch. Use the following steps to turn on trusted access in the CloudWatch console.

To turn on trusted access

1. Sign in to the console with your organization's management account.
2. In the CloudWatch console, in the navigation pane, choose **Settings**.
3. Choose the **Organizations** tab.
4. In **Organizational Management Settings**, choose **Turn on**. The **Enable trusted access** page appears.
5. To review the role policy, choose **View permission** details to see the role policy.
6. Choose **Enable trusted access**.

Now, as CloudWatch discovers resources, it automatically updates information about accounts that you have permission to access the resources for in Network Flow Monitor.

Register a delegated administrator account

As a best practice with AWS Organizations, the management account for your organization should register a member account as a delegated administrator account for CloudWatch. After you register a delegated administrator account in CloudWatch, members of your organization can sign in with the delegated administrator account to monitor the network performance for resources in multiple accounts in Network Flow Monitor.

Using the delegated administrator account, you can add multiple accounts for your network observability scope in Network Flow Monitor. Although the management account can also create a scope that includes multiple accounts, we recommend that you follow the best practices for AWS Organizations and use a delegated administrator account for adding multiple accounts in Network Flow Monitor. For member accounts that are not the delegated administrator account, the scope is limited to the signed-in account, which is automatically set for the scope.

A delegated administrator account for Organizations is a member account that shares administrator access for service-managed permissions. The account that you choose to register as a delegated administrator account must be a member account in your organization. A delegated administrator account for your organization can be used outside of CloudWatch, so make sure that you understand this account type before following this procedure. For more information, see Amazon CloudWatch and AWS Organizations in the AWS Organizations User Guide.

To register a delegated administrator account

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**.
3. Choose the **Organization** tab.
4. Choose **Register delegated administrator**.
5. In the **Register delegated administrator** window, in the **Delegated administrator account ID** field, enter the 12-digit Organization member account ID.
6. Choose **Register delegated administrator**. A confirmation message appears indicating the account was registered successfully. The **Organization Settings** page appears. To see information about the delegated administrator account, hover over the number next to **Delegated administrators**.

To remove or change the delegated administrator account, deregister the account first. For more information, see Deregistering a delegated administrator account.

Add multiple accounts to your scope

To add accounts to your Network Flow Monitor scope, sign in with the delegated administrator account. (You can add accounts to a scope if you're signed in with the management account. However, it's a best practice in AWS Organizations to use the delegated administrator account to work with resources.)

After you sign in, follow the steps to initialize Network Flow Monitor, a process which authorizes the required service-linked role permissions, lets you set the scope for your network observability by adding accounts, and then creates an initial topology for the accounts in the scope you've set. The account that you sign in with—in this case, the delegated administrator account—is automatically included in your Network Flow Monitor scope.

To initialize Network Flow Monitor with multiple accounts in your scope

1. Sign in to the console with your organization's delegated administrator account.
2. In the navigation pane for the CloudWatch console, under **Network Monitoring**, choose **Flow monitors**.
3. Under **Getting started with Network Flow Monitor**, in Step 1, choose **Start initialization**.
4. On the **Network Flow Monitor** page, under **Add accounts**, choose **Add**. The account that you're signed in with is automatically included in the scope and already appears in the **Accounts in scope** table as **(this account)**.
5. On the **Add accounts** dialog page, optionally filter the accounts, and then select up to 99 additional accounts to add to your scope. The maximum number of accounts in a scope is 100.
6. Choose **Add**.
7. Choose **Initialize Network Flow Monitor**. Network Flow Monitor adds the required service-linked role permissions, creates a scope that includes all the accounts you specified, and then creates an initial topology of the resources in the accounts in your scope.

To add or remove accounts for your scope after you've already initialized Network Flow Monitor, follow the steps here.

Be aware that after you make a change to your scope, either to add or delete accounts, you must wait about 20 minutes before you can make another change to the scope. This delay is because Network Flow Monitor requires a brief period of time to update its topology information.

To add or remove accounts for your scope

1. Sign in to the console with your organization's delegated administrator account.
2. In the navigation pane for the CloudWatch console, under **Network Monitoring**, choose **Flow monitors**.
3. Under **Monitors**, select a monitor.
4. On the **Monitor details** tab, under **Accounts in scope**, choose **Add** or **Delete**.
5. Select accounts to add to your scope, up to a total of 100 accounts, or select accounts to delete.
6. Complete the steps in the confirmation dialog.

Set up permissions for multi-account resource access (console only)

If you plan to create monitors for network flows from the console, a specific policy is required for each member account in your scope. This policy enables you to view resources from other accounts when you add local and remote resources to a monitor.

For each of the account in your scope, create a role, **NetworkFlowMonitorAccountResourceAccess**, and attach the **AmazonEC2ReadOnlyAccess** policy. To see permission details for the policy, see AmazonEC2ReadOnlyAccess in the AWS Managed Policy Reference Guide.

If you want to view Amazon EKS cluster resources from other accounts in the console, you must also add the following permissions to the **NetworkFlowMonitorAccountResourceAccess** role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EKSReadAccess",
            "Effect": "Allow",
            "Action": [
                "eks:ListClusters",
                "eks:DescribeCluster",
                "eks:ListNodegroups",
                "eks:DescribeNodegroup",
```

```
                "eks:ListFargateProfiles",
                "eks:DescribeFargateProfile",
                "eks:ListAddons",
                "eks:DescribeAddon",
                "eks:AccessKubernetesApi"
            ],
            "Resource": "*"
        }
    ]
}
```

This policy is in addition to the policy that you must add to each instance so that the Network Flow Monitor agent can send performance metrics from the instance to the Network Flow Monitor ingestion backend server. For more information about requirements for agents, see Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances.

The following procedure provides a summary of the steps to create the required role for accessing resources in your scope in the Network Flow Monitor console. For general guidance on how to create a role in IAM, see Create a role to give permissions to an IAM user in the AWS Identity and Access Management User Guide.

To create a role for resource access in the Network Flow Monitor console

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane of the console, choose **Roles**, and then choose **Create role**.
3. Specify the **AWS account** trusted entity. This trusted entity type enables principals in other AWS accounts to assume the role and access resources in other accounts.
4. Choose **Next**.
5. In the list of AWS managed policies, choose the **AmazonEC2ReadOnlyAccess** policy.
6. Choose **Next**.
7. For role name, enter **NetworkFlowMonitorAccountResourceAccess**.
8. Review the role, and then choose **Create role**.

Install agents on instances

To track network performance with Network Flow Monitor, you must initialize the service. You must also install Network Flow Monitor agents on your workload's EC2 instances and add permissions for the agents to send networking performance metrics to Network Flow Monitor.

After you install the agents, wait a short period of time (about 20 minutes), for data to begin being sent to the Network Flow Monitor backend. Then, you can view network performance metrics, on the **Workload insights** tab, and also create monitors, to view detailed information.

For example, you can view the top contributor performance metrics for data transferred and retransmission timeouts, for network flows between your local and remote resources, collected by Network Flow Monitor agents. By viewing and analyzing these metrics, you can choose specific flows that you want to see more details for and track more closely with a monitor. By creating a monitor for specific flows, you can see detailed information about them, including metrics, sorted by the top contributors for each metric type and network paths for each network flow.

With a monitor, Network Flow Monitor also provides a network health indicator (NHI), which you can use to see if there have been AWS network impairments for network flows that you're tracking in the monitor, during a time period that you've selected. That information can help you decide where to focus your network troubleshooting efforts.

For more information, and instructions for how to install agents, see Install Network Flow Monitor agents on EC2 and self-managed Kubernetes instances.

Monitor and analyze network flows in Network Flow Monitor

With Network Flow Monitor, you can learn about network flows and performance for traffic that you're monitoring with your scope. Start by reviewing top contributors information, by each metric type, in the **Workload insights** tab. Then, create monitors so that you can explore network performance in detail, over different time frames, for network flows that you select in **Workload insights**.

After you initialize Network Flow Monitor and install agents on your instances, Network Flow Monitor generates performance metrics for the network flows from those instances. Review the sections in this chapter to learn more about how to use Network Flow Monitor to evaluate network performance in Network Flow Monitor, create monitors for more in-depth information, and learn about the specific metrics provided by Network Flow Monitor.

The steps provided in these sections use the AWS Management Console. You can also use Network Flow Monitor API operations with the AWS Command Line Interface (AWS CLI) or AWS SDKs to review workload insights and top contributors metrics, and to create and configure a monitor. For more information, see the following resources:

- If you plan to work with Network Flow Monitor with the CLI, see Examples of using the CLI with Network Flow Monitor.

- For detailed information about working with Network Flow Monitor API operations, see the Network Flow Monitor API Reference Guide.

Contents

- Evaluate network flows with workload insights
- Create and work with monitors in Network Flow Monitor
- Monitor and analyze network flows with a Network Flow Monitor monitor
- Delete scope for Network Flow Monitor

Evaluate network flows with workload insights

Network Flow Monitor provides workload insights about the network flows in the scope that you enable monitoring for. By surfacing top contributor network flows for each metric type, you can see which flows are potentially experiencing issues. You can view these top contributor metrics in the console on the **Workload insights** tab. In Network Flow Monitor, top contributors are network flows that have the highest values for each network performance metric.

Top contributors

On the **Workload insights** page in the Network Flow Monitor console, Network Flow Monitor displays the top contributor network performance statistics for the network flows between all the resources in your monitoring scope where you've deployed agents.

To compile lists of top contributors, Network Flow Monitor determines the network flows in your scope that have the highest values for retransmissions, retransmission timeouts, and data transferred. These network flows are the *top contributors* for each metric type.

For workload insights, the top contributors are determined for all the network flows that you're receiving performance information for; that is, network flows for all of the resources in your scope with Network Flow Monitor agents installed.

Network flow classifications

Network Flow Monitor classifies metrics into designated local-remote categories. Metrics are grouped into two types of flows:

- **Network Health Indicator (NHI) Flows:** Flows which contribute to Network Health Indicator (NHI) calculations:

- Between AZs (`INTER_AZ`). Always within the same VPC.
- Within AZs (`INTRA_AZ`). Always within the same VPC.
- Between VPCs (`INTER_VPC`). Crosses the boundary between VPCs.
- Between Regions (`INTER_REGION`). This means performance for network flows between resources in your Region and the edge of another Region.
- Toward Amazon S3 buckets (`AMAZON_S3`)
- Toward Amazon DynamoDB (`AMAZON_DYNAMODB`)

- **Non Network Health Indicator (NHI) Flows:** Flows which do not contribute to Network Health Indicator (NHI) calcuations:
  - Toward the Internet (`INTERNET`). Flows that traverse an Internet Gateway and end on the public Internet.
  - Towards AWS Services (`AWS_SERVICE`). Flows that end at an AWS service that is not fully monitored (like CloudFront or API Gateway).
  - Towards a Transit Gateway (`TRANSIT_GATEWAY`). Flows in this classification are flows that arrive at a Transit Gateway, but the final destination of the flow is unknown.
  - Towards a local zone (`LOCAL_ZONE`). Flows that start or end in a local zone
  - Any other flow that cannot be otherwise classified (`UNCLASSIFIED`).

Performance metrics

Performance metrics on **Workload insights** are shown in separate tables for the following metric type: Retransmissions, retransmission timeouts, and data transferred. The data provided is for the top contributors for each type. Note that after you first install Network Flow Monitor agents, there is a waiting period (about 20 minutes) before you can view performance metrics, while agents gather and send data to the Network Flow Monitor backend.

Monitor specific network flows

As you review performance metrics, when you see specific resources or network flows that you want to explore more details for, you can create a monitor that includes just those flows.

With a monitor, you can track specific groups of network flows over a period of time, to gain insights into one aspect of your workload. You can also get helpful troubleshooting information, such as checking the network health indicator (NHI) to see if issues you see are caused by AWS impairments.

For more information, see Create and work with monitors in Network Flow Monitor

Multi-account coverage

To work with multiple accounts in Network Flow Monitor, you must configure AWS Organizations integration with CloudWatch. By configuring Organizations, you can add accounts to the scope for your Network Flow Monitor coverage. Then, if you have multiple accounts in your scope that each have resources that you want to monitor network flows between, you can specify a scope that includes all of the accounts, and then view performance metrics gathered by the Network Flow Monitor agents that you've installed on your resources. For more information, see Initialize Network Flow Monitor for multi-account monitoring.

Create and work with monitors in Network Flow Monitor

You create a monitor to see details about the network performance for one or several network flows for a workload. For each monitor, Network Flow Monitor publishes end-to-end performance metrics and a network health indicator (NHI), and generates network paths of individual network flows. After you create a monitor, you can view information provided by the monitor in the console on the **Monitors** tab.

Flow monitors can help you to assess network performance issues that are impacting your workloads, including impairments within an AWS Region and issues on the AWS global network between a local and a remote Region. The network health indicator (NHI) provided by the monitor also captures the health of the AWS global network on your workload's network paths between Regions. This helps you to quickly identify whether impairments in a local Region, in the AWS global network, or in the remote Region are affecting your workloads.

For remote Regions, monitors can provide network visibility for flows to the Region's public IP address, and for private traffic flowing to a remote Region over VPC peering or Transit Gateway peering.

After you create a monitor, you can edit the monitor to make changes (except change the monitor name) or delete the monitor, at any time.

The following sections includes procedures for creating, editing, and deleting monitors in the Network Flow Monitor console.

Contents

- Create a monitor in Network Flow Monitor
- Edit a monitor in Network Flow Monitor
- Delete a monitor in Network Flow Monitor

Create a monitor in Network Flow Monitor

As you review top contributors in the **Workload insights** tab, if you see one or several network flows that you want to follow over time, or that you want more details about, you can create a monitor directly from **Workload insights**. This simplifies the process for creating a monitor for specific network flows.

Or, if you know specific network flows that you want to track with a monitor, such as looking at performance information for all network flows to another AWS Region, you can use the **Create monitor** wizard to create a monitor from scratch. When you create a monitor this way, you specify all of the local and remote resources that define the network flows that you want to monitor.

For specific procedures, see the following sections:

- Create a monitor by specifying network flows
- Create a monitor by specifying local and remote resources

Create a monitor by specifying network flows

To create a monitor by selecting network flows, start on the **Workload insights** tab. Select one or more network flows in one of the tables, in a single Region, and then, choose to create a monitor with those flows.

When you create a monitor in this way, the **Create monitor** wizard pre-populates local and remote resources for you and displays them in a modal dialog. You can choose to create a monitor with those resources, or edit the selection of local or remote resources to add or remove resources to include.

By reviewing the top contributors on **Workload insights** on an ongoing basis, you can regularly evaluate if you have the monitors that you need, or if creating new monitors would be helpful.

> **Important**
>
> These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To create a monitor from Workload insights

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. Choose **Workload insights**.
4. In one of the **Top contributors** tables, select one or more network flows and then choose **Create monitor**.
5. In the modal window that opens, you can edit the resources that define the network flows that you chose, or choose **Create monitor**.

Create a monitor by specifying local and remote resources

You can create a monitor at any time for specific local and remote resources that define network flows that you want to see details for.

For example, you might want to create a monitor for one of the following scenarios:

- A monitor that includes network flows for a specific VPC in a local Region to another VPC in the same Region. (Note that you can't select a specific resource, such as a VPC, as a network flow endpoint - that is, the remote resource - in another Region.)
  - For local resource, choose **Specific resources in *Region***. Then, choose **VPC and subnets**, and then, in the table, select a specific VPC.
  - For remote resource, do the same: choose **Specific resources in *Region***, then, choose **VPC and subnets**, and finally, select a specific VPC.
- A monitor that includes all network flows from your workload in a local Region to a specific Availability Zone.
  - For local resource, choose **Everywhere in *Region***
  - For remote resource, choose **Availability Zone**, and then choose a specific AZ
- A monitor that includes all network flows for your workload within a local Region.
  - For local resource, choose **Everywhere in *Region***
  - For remote resource, choose **Everywhere in *Region***
- A monitor that includes all network flows for your workload from a local Region to the edge of another Region.
  - For local resource, choose **Everywhere in *Region***
  - For remote resource, choose **Another Region**, and then choose the remote Region

> **Important**
>
> These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To create a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. From the Network Flow Monitor page, select the **Monitors** tab, and then choose **Create monitor**.
4. For **Monitor name**, enter the name that you want to use for the monitor. You can't change this name later.
5. Choose **Next**.
6. Select the local resources (one or more) for the network flows that you want to monitor.
   - To monitor network flows from all resources in your Region, choose **Everywhere in *Region***.
   - To choose specific local resources to monitor flows from, choose **Specific resources in *Region***. Then, under **Add resources**, choose **Availability Zones**, **EKS clusters**, or **VPCs and subnets**, and then choose resources to add.
7. Choose **Next**.
8. Select the remote resources (one or more) for the network flows that you want to monitor.
   - To monitor network flows to all resources in your Region, choose **Everywhere in *Region***.
   - To monitor flows from specific remote resources, choose **Specific resources in *Region***. Under **Add resources**, select **VPCs and subnets**, **Availability Zones**, or **AWS services**, and then choose the resources to add.
   - To monitor network flows to the edge of another Region, choose **Another Region**.
9. Choose **Next**.
10. Review your choices to confirm the network flows to monitor, or edit the options to make changes.
11. Choose **Create monitor**.

After you create a monitor, you can edit or delete the monitor at any time to add or remove network flows. Select a monitor, and then choose **Edit** or **Delete**. Note that you can't change the name of a monitor.

To view the Network Flow Monitor dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Network monitoring**, then **Flow monitors**.

   The **Monitors** tab displays a list of the monitors that you have created.

To see more information about a specific monitor, choose a monitor.

Edit a monitor in Network Flow Monitor

You can edit a monitor at any time, to add or remove network flows.

Note that you can't change the name of a monitor after you create it.

> **Important**
>
> These steps are designed to be completed all at once. You won't be able to save any in-process work to continue later.

To edit a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Monitors** tab, select a monitor, and then under the **Actions** menu, choose **Edit**.
4. Select the local or remote resources that you want add or remove for the monitor. If you have multiple accounts in your scope, specify the account where the resources are located, and then choose resources.
5. When you're finished updating the monitor, choose **Next**, to review and confirm the network flows to monitor.
6. Choose **Save monitor**.

Delete a monitor in Network Flow Monitor

To delete a monitor in Network Flow Monitor, follow the steps here.

To delete a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Monitors** tab, select a monitor, and then under the **Actions** menu, choose **Delete**.
4. In the dialog that appears, enter confirmation text, and then choose **Delete**.

Monitor and analyze network flows with a Network Flow Monitor monitor

Network Flow Monitor data and graphs help you to visualize and track network issues. You can create monitors to see detailed information about specific network segments for your AWS workloads, including a view of the network path for individual network flows. After you create one or more monitors in Network Flow Monitor, you can observe performance and metrics, and explore historical data, to find anomalies.

To see the information provided by a monitor, on the **Monitors** tab, choose a monitor in the **Monitors** table. Then, choose one of the following tabs for more information: **Overview**, **Historical explorer**, or **Monitor details**.

Overview tab

On the **Overview** tab, you can review the following, for time periods that you specify. To see a broader or narrower range of historical information, including the NHI and traffic summary data, adjust the time period selection at the top of the page.

- **Network health indicator (NHI):** NHI alerts you to whether there were AWS network issues for one or more of the network flows tracked by your monitor, during the time frame that you've selected for viewing performance metrics. NHI is a binary value, that is, 100 or 0, which is shown in the console as **Degraded** or **Healthy**.
  - NHI is shown as **Degraded** if there were issues with the portion of the AWS network that any network flow in the monitor traversed, at any time during the time frame that you select.
  - Otherwise, NHI is shown as **Healthy**.

If the NHI is **Degraded**, you can view the **Network health indicator** bar graph for more information. The graph shows you when, during the selected time frame, there were AWS network issues for the network flows tracked by your monitor.

- **Traffic summary:** Observe overall metrics for the flows tracked by this monitor, for the time period that you've selected. You can see average round-trip time, sums (totals) of transmission timeouts and retransmissions, and the average amount of data transferred for the flows in the monitor. Be aware that RTT data can be sparse because RTT is not always calculated.

Historical explorer tab

On the **Historical explorer** tab, you can dive deep into information about specific flows. You can review metrics and network paths for top contributor network flows for specified time frames. In the tables of metrics, you can filter the data by different categories of flows, such as flows between Availability Zones (`INTER_AZ`).

- **Metrics:** View detailed information for the top contributors for each metric type that Network Flow Monitor aggregates data for. Separate tables of top contributors are provided for retransmission timeouts, retransmissions, and data transferred.
- **Network paths:** To get an idea about where anomalies are occurring, you can view the network path of a network flow. When you choose a specific metric in a metrics table, the network path for that flow is displayed below the table.

Monitor details tab

On the **Monitor details** tab, you can see details about the monitor, including the monitor state, the ARN, when it was created and last updated, and the flows that are included.

You can choose to edit or delete a monitor from any page on the **Monitors** tab.

As part of your regular use of Network Flow Monitor, we recommend that you periodically review the data on the **Workload insights** page to determine if there are new flows that show metrics anomalies that you want to track more closely over time. When you see a set of flows on the **Workload insights** page that you want to see details about, select the flows and create a monitor for them.

Delete scope for Network Flow Monitor

If you decide that you no longer want to monitor network flows using Network Flow Monitor, you can delete your Network Flow Monitor scope. When you delete your scope, you can no longer see network performance information.

Before you can delete your scope, you must delete all monitors. Be aware that it can take about 15 minutes to complete removing monitors after you request to delete them. For more information, see Delete a monitor in Network Flow Monitor.

To delete your scope

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Flow monitors**.
3. On the **Settings** tab, choose **Delete scope**.
4. In the dialog box, enter confirmation text, and then choose **Delete scope**.

View Network Flow Monitor metrics in CloudWatch

Network Flow Monitor publishes the following network flow performance metrics to your account: round-trip time, TCP retransmissions, TCP retransmission timeouts, data transferred, and network health indicator. You can view these metrics in CloudWatch Metrics in the Amazon CloudWatch console.

To find all metrics for your monitor, in the CloudWatch Metrics dashboard, see the custom namespace `AWS/NetworkFlowMonitor`. Metrics are aggregated for each monitor that is deployed and active.

Network Flow Monitor provides the following metrics. Be aware that RoundTripTime data can be sparse, as this metric is not always calculated.

Metric	Description
DataTransferred	The number of bytes transferred for all flows for a monitor.
Retransmissions	Total number of retransmissions for a monitor. Retransmissions occur when the sender needs

Metric	Description
	to resend packets that have been either damaged or lost.
Timeouts	Total retransmission timeouts for a monitor. This occurs when the sender is missing too many acknowledgments, and therefore decides to take a time out and stop sending altogether.
RoundTripTime	Average round-trip time for network flows for a monitor. This metric, measured in microseconds, is a measure of performance. It records the time it takes for traffic to be transmitted from a local resource to a remote IP address, and for the associated response to be received. The time is an average over the aggregation period. Data can be sparse, as this metric is not always calculated.
HealthIndicator	Network health indicator (NHI) for a monitor overall. Network health indicator (NHI) is a value that surfaces an AWS network impairment. The NHI value is 100 (degraded) if there was an AWS network issue during a specified time frame. It's set to 0 (healthy) if no AWS network issues were detected. Observing the NHI can help you to prioritize troubleshooting for either your workload or the AWS network.

Create alarms with Network Flow Monitor

You can create Amazon CloudWatch alarms based on Network Flow Monitor metrics, just as you can for other CloudWatch metrics.

For example, you can create an alarm based on the Network Flow Monitor metric `Retransmissions`, and configure it to send a notification when the metric is higher than a value that you choose. You configure alarms for Network Flow Monitor metrics following the same guidelines as for other CloudWatch metrics.

Following are example Network Flow Monitor metrics that you might choose to create an alarm for:

- **Retransmissions**
- **Timeouts**
- **RoundTripTime**

To see all the metrics available for Network Flow Monitor see Create a CloudWatch alarm based on a static threshold.

The following procedure provides an example of setting an alarm on **Retransmissions** by navigating to the metric in the CloudWatch dashboard. Then, you follow the standard CloudWatch steps to create an alarm based on a threshold that you choose, and set up a notification or choose other options.

To create an alarm for Retransmissions in CloudWatch Metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, and then choose **Classic metrics**.
3. Filter for Network Flow Monitor by choosing `AWS/NetworkFlowMonitor`.
4. Choose **MeasurementSource, MonitorName**.
5. In the list, select **Retransmissions**.
6. On the **GraphedMetrics** tab, under **Actions**, choose the bell icon to create an alarm based on a static threshold.

Now, follow the standard CloudWatch steps to choose options for the alarm. For example, you can choose to be notified by an Amazon SNS message if **Retransmissions** is below a specific threshold number. Alternatively, or in addition, you can add the alarm to a dashboard.

Keep in mind the following:

- Network Flow Monitor metrics are typically aggregated and sent to the Network Flow Monitor backend every 30 seconds, with a 5 second potential jitter (in other words, 25 to 35 seconds).
- When you create an alarm based on Network Flow Monitor metrics, make sure that you take into account the short delay before publication when you set an alarm's lookback period. We recommend that you configure **Evaluation Periods** with lookback period that is a minimum of 25 minutes.

For more information about options when you create a CloudWatch alarm, see Create a CloudWatch alarm based on a static threshold.

AWS CloudTrail for Network Flow Monitor

Monitoring a service is an important part of maintaining reliability, availability, and performance. of Network Flow Monitor and your other AWS solutions. *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

For more information about Network Flow Monitor CloudTrail logging, see Network Flow Monitor in CloudTrail.

Troubleshoot issues in Network Flow Monitor

This section provides guidance for troubleshooting errors with Network Flow Monitor, including solving issues with installing agents.

Troubleshoot issues in EKS agents installation

When you try to upgrade the AWS Network Flow Monitor Agent add-on for EKS from v1.0.0 to v1.0.1 in AWS Management Console, you might receive the following error message:

"Service account `aws-network-flow-monitoring-agent-service-account` in pod identity configuration is not supported for addon `aws-network-flow-monitoring-agent`."

This error is returned because a resource was renamed. The EKS add-on v1.0.1 changes the service account name from `aws-network-flow-monitoring-agent-service-account` to `aws-network-flow-monitor-agent-service-account`.

Then, if **Not set** is not selected in the console, the pod identity association is not reset to the new resource name.

To fix this issue, do the following when you upgrade to the new version by using the console:

1. Under **Pod Identity IAM role for service account**, select **Not set**.
2. Select **New version (v1.0.1)**.
3. Select **Upgrade**.
4. Choose **Save changes**.

Data security and data protection in Network Flow Monitor

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Network Flow Monitor, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Network Flow Monitor. The following topics show you how to configure Network Flow Monitor to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Network Flow Monitor resources.

Topics

- Data protection in Network Flow Monitor
- Infrastructure Security in Network Flow Monitor

- Identity and Access Management for Network Flow Monitor

Data protection in Network Flow Monitor

The AWS shared responsibility model applies to data protection in Network Flow Monitor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Network Flow Monitor or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Infrastructure Security in Network Flow Monitor

As a managed service, Network Flow Monitor is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Network Flow Monitor through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Identity and Access Management for Network Flow Monitor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Network Flow Monitor resources. IAM is an AWS service that you can use with no additional charge.

Contents

- How Network Flow Monitor works with IAM
- AWS managed policies for Network Flow Monitor
- Service-linked roles for Network Flow Monitor

How Network Flow Monitor works with IAM

Before you use IAM to manage access to Network Flow Monitor, learn what IAM features are available to use with Network Flow Monitor.

To see tables showing a similar high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

IAM feature	Network Flow Monitor support
Identity-based policies	Yes

IAM feature	Network Flow Monitor support
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Network Flow Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Resource-based policies within Network Flow Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that

support resource-based policies, service administrators can use them to control access to a specific resource.

Policy actions for Network Flow Monitor

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Network Flow Monitor actions, see Actions defined by Network Flow Monitor in the *Service Authorization Reference*.

Policy actions in Network Flow Monitor use the following prefix before the action:

```
networkflowmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "networkflowmonitor:action1",
      "networkflowmonitor:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "networkflowmonitor:Describe*"
```

Policy resources for Network Flow Monitor

Supports policy resources: Yes

In the *Service Authorization Reference*, you can see the following information related to Network Flow Monitor:

- To see a list of Network Flow Monitor resource types and their ARNs, see Resources defined by Network Flow Monitor.
- To learn the actions that you can specify with the ARN of each resource, see Actions defined by Network Flow Monitor.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Network Flow Monitor

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Network Flow Monitor condition keys, see Condition keys for Network Flow Monitor in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Network Flow Monitor.

ACLs in Network Flow Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Network Flow Monitor

Supports ABAC (tags in policies): Yes

Network Flow Monitor has *partial* support for tags in policies. It supports tagging for one resource, monitors.

To use tags with Network Flow Monitor, use the AWS Command Line Interface or an AWS SDK. Tagging for Network Flow Monitor is not supported with the AWS Management Console.

To learn more about using tags in policies in general, review the following information.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Network Flow Monitor

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Network Flow Monitor

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Network Flow Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

Service-linked role for Network Flow Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For more information about the service-linked role for Network Flow Monitor, see Service-linked roles for Network Flow Monitor.

For details about creating or managing service-linked roles in general in AWS, see AWS services that work with IAM. Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

AWS managed policies for Network Flow Monitor

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users,

groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: CloudWatchNetworkFlowMonitorServiceRolePolicy

You can't attach `CloudWatchNetworkFlowMonitorServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role named **AWSServiceRoleForNetworkFlowMonitor**, which publishes network telemetry aggregation results, collected by Network Flow Monitor agents, to CloudWatch. It also allows the service to use AWS Organizations to get information for multi-account scenarios.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorServiceRolePolicy in the *AWS Managed Policy Reference*.

For more information, see Service-linked roles for Network Flow Monitor.

AWS managed policy: CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy

You can't attach `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role named **AWSServiceRoleForNetworkFlowMonitor_Topology**. Using these permissions, as well as internal meta data information gathering (for performance efficiencies), this service-linked role gathers meta data about resource network configurations, such as describing route tables and gateways, for resources that this service monitors network traffic for. This meta data enables Network Flow Monitor to generate topology snapshots of the resources. When there is network degradation, Network Flow Monitor uses the topologies to provide insights into the location of issues in the network and to help determine attribution for issues.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy in the *AWS Managed Policy Reference*.

For more information, see Service-linked roles for Network Flow Monitor.

AWS managed policy: CloudWatchNetworkFlowMonitorAgentPublishPolicy

You can use this policy in IAM roles that are attached to Amazon EC2 and Amazon EKS instance resources to send telemetry reports (metrics) to a Network Flow Monitor endpoint.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorAgentPublishPolicy in the *AWS Managed Policy Reference*.

Updates to the Network Flow Monitor service-linked roles

For updates to the AWS managed policies for the Network Flow Monitor service-linked roles, see the AWS managed policies updates table for CloudWatch. You can also subscribe to automatic RSS alerts on the CloudWatch Document history page.

Service-linked roles for Network Flow Monitor

Network Flow Monitor uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Network Flow Monitor. The service-linked role is predefined by Network Flow Monitor and includes all the permissions that the service requires to call other AWS services on your behalf.

Network Flow Monitor defines the permissions of the service-linked roles, and unless defined otherwise, only Network Flow Monitor can assume the roles. The defined permissions include the trust policies and the permissions policies, and the permissions policies cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This restriction protects your Network Flow Monitor resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Network Flow Monitor

Network Flow Monitor uses the following service-linked roles:

- **AWSServiceRoleForNetworkFlowMonitor**
- **AWSServiceRoleForNetworkFlowMonitor_Topology**

Service-linked role permissions for AWSServiceRoleForNetworkFlowMonitor

Network Flow Monitor uses the service-linked role named **AWSServiceRoleForNetworkFlowMonitor**. This role allows Network Flow Monitor to publish

CloudWatch aggregated telemetry metrics gathered for network traffic between instances, and between instances and AWS locations. It also allows the service to use AWS Organizations to get information for multi-account scenarios.

This service-linked role uses the managed policy `CloudWatchNetworkFlowMonitorServiceRolePolicy`.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorServiceRolePolicy in the *AWS Managed Policy Reference*.

The **AWSServiceRoleForNetworkFlowMonitor** service-linked role trusts the following service to assume the role:

- `networkflowmonitor.amazonaws.com`

Service-linked role permissions for AWSServiceRoleForNetworkFlowMonitor_Topology

Network Flow Monitor uses the service-linked role named **AWSServiceRoleForNetworkFlowMonitor_Topology**. This role allows Network Flow Monitor to generate a topology snapshot of the resources that you use with Network Flow Monitor.

This service-linked role uses the managed policy `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy`.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy in the *AWS Managed Policy Reference*.

The **AWSServiceRoleForNetworkFlowMonitor_Topology** service-linked role trusts the following service to assume the role:

- `topology.networkflowmonitor.amazonaws.com`

Creating a service-linked role for Network Flow Monitor

You do not need to manually create the service-linked roles for Network Flow Monitor. The first time that you initialize Network Flow Monitor, Network Flow Monitor creates **AWSServiceRoleForNetworkFlowMonitor** and **AWSServiceRoleForNetworkFlowMonitor_Topology** for you.

For more information, see Creating a service-linked role in the *IAM User Guide*.

Editing a service-linked role for Network Flow Monitor

After Network Flow Monitor creates a service-linked role in your account, you cannot change the name of the role because various entities might reference the role. You can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Network Flow Monitor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete the role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for the service-linked role before you can manually delete it.

> **Note**
>
> If the Network Flow Monitor service is using the role when you try to delete it, then the deletion might fail. If that happens, wait for a few minutes and then try again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForNetworkFlowMonitor** or the **AWSServiceRoleForNetworkFlowMonitor_Topology** service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Updates to the Network Flow Monitor service-linked role

For updates to `CloudWatchNetworkFlowMonitorServiceRolePolicy` or `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy`, the AWS managed policies for the Network Flow Monitor service-linked roles, see CloudWatch updates to AWS managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Using Internet Monitor

Internet Monitor provides visibility into how internet issues impact the performance and availability between your applications hosted on AWS and your end users. It can reduce the

time it takes for you to diagnose internet issues from days to minutes. Internet Monitor uses the connectivity data that AWS captures from its global networking footprint to calculate a baseline of performance and availability for internet-facing traffic. This is the same data that AWS uses to monitor internet uptime and availability. With those measurements as a baseline, Internet Monitor raises awareness for you when there are significant problems for your end users (clients) in the different geographic locations where your application runs.

In the Amazon CloudWatch console, you can see a global view of traffic patterns and health events, and easily drill down into information about events, at different geographic granularities (locations). You can clearly visualize impact, and pinpoint the client locations and networks (ASNs, typically internet service providers or ISPs) that are affected. If Internet Monitor determines that an internet availability or performance issue is caused by a specific ASN or by the AWS network, it provides that information.

To get started, create a monitor that includes one or more resources, so Internet Monitor can create a traffic profile for your AWS application. Then, view information in the Internet Monitor dashboard to visualize data and get insights and suggestions about your application's internet traffic.

For information about Regional support, pricing, how Internet Monitor works, and other overview content, see What is Internet Monitor?. To begin working with Internet Monitor, see Getting started with Internet Monitor using the console.

What is Internet Monitor?

With Internet Monitor, you can monitor your application's internet performance and availability, so that you can visualize data and get insights and suggestions about your AWS application's internet traffic. You can also get suggestions for ways to reduce latency for your application, by using different Regions or AWS services, like Amazon CloudFront.

Key features of Internet Monitor

- Internet Monitor suggests insights and recommendations that can help you improve your end users' experience. You can explore, in near real-time, how to improve the projected latency of your application by switching to use other services, or by rerouting traffic to your workload through different AWS Regions.
- Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each AWS Region and edge location. With the Internet Monitor dashboard,

you can quickly identify what's impacting your application's performance and availability, so that you can track down and address issues.

- Internet Monitor also publishes internet measurements to CloudWatch Logs and CloudWatch Metrics, to support using CloudWatch tools to explore data for city-networks that are specific to your monitored application traffic. Optionally, you can also publish internet measurements to Amazon S3.
- Internet Monitor sends health events to Amazon EventBridge so that you can set up notifications. Both overall (global) health events and local health events are published to EventBridge. If an issue is caused by the AWS network, you also automatically receive an AWS Health Dashboard notification with the steps that AWS is taking to mitigate the problem.

How to use Internet Monitor

To use Internet Monitor, you create a *monitor* and associate your application's resources with it—VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories—to enable Internet Monitor to know where your application's internet-facing traffic is. Internet Monitor then publishes internet measurements from AWS that are specific to the *city-networks*, that is, the client locations and ASNs (typically internet service providers or ISPs), where clients access your application. For more information, see How Internet Monitor works. To begin working with Internet Monitor, see Getting started with Internet Monitor using the console.

Contents

- Supported AWS Regions for Internet Monitor
- Components and terms for Internet Monitor
- How Internet Monitor works
- Internet Monitor example use cases
- Global internet weather map in Internet Monitor
- Internet Monitor cross-account observability
- Pricing for Internet Monitor

Supported AWS Regions for Internet Monitor

The AWS Regions and AWS Local Zones where Amazon CloudWatch Internet Monitor is supported are listed in this section. For more information about Regions that Internet Monitor is supported in,

including opt-in Regions, see Amazon CloudWatch Internet Monitor endpoints and quotas in the *Amazon Web Services General Reference*.

Note that Internet Monitor stores data for a monitor in only the AWS Region in which you create the monitor, although a monitor can include resources in multiple Regions.

Region name (Opt-in support)	Region
Africa (Cape Town)	`af-south-1`
Asia Pacific (Hong Kong)	`ap-east-1`
Asia Pacific (Hyderabad)	`ap-south-2`
Asia Pacific (Jakarta)	`ap-southeast-3`
Asia Pacific (Melbourne)	`ap-southeast-4`
Europe (Milan)	`eu-south-1`
Europe (Spain)	`eu-south-2`
Europe (Zurich)	`eu-central-2`
Middle East (Bahrain)	`me-south-1`
Middle East (UAE)	`me-central-1`

Region name (Default support)	Region
US East (Ohio)	`us-east-2`
US East (N. Virginia)	`us-east-1`
US West (N. California)	`us-west-1`
US West (Oregon)	`us-west-2`
Asia Pacific (Mumbai)	`ap-south-1`

Region name (Default support)	Region
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

For Local Zones support, you must enable the Local Zone and attach it to the VPC that you want to monitor internet traffic for. Internet Monitor does not support Local Zones for other resources types. The Local Zones that are supported are listed in the following table.

Local Zone	Parent Region	Type
us-east-1-dfw-2a	us-east-1	Availability Zone
us-east-1-mia-2a	us-east-1	Availability Zone
us-east-1-qro-1a	us-east-1	Frontier Zone
us-east-1-lim-1a	us-east-1	Frontier Zone

Local Zone	Parent Region	Type
us-east-1-atl-2a	us-east-1	Availability Zone
us-east-1-bue-1a	us-east-1	Frontier Zone
us-east-1-mci-1a	us-east-1	Frontier Zone
us-west-2-lax-1a	us-west-2	Availability Zone
us-west-2-lax-1b	us-west-2	Availability Zone
af-south-1-los-1a	af-south-1	Frontier Zone

Components and terms for Internet Monitor

Internet Monitor uses or references the following concepts.

Monitor

A monitor includes the resources for a single application that you want to view internet performance and availability measurements for, and that you want to get health event alerts about. When you create a monitor for an application, you add resources for the application to define the cities (locations) for Internet Monitor to monitor. Internet Monitor uses the traffic patterns from the application resources that you add so that it can publish internet performance and availability measurements specific to just the locations and ASNs (typically, internet service providers or ISPs) that communicate with your application. In other words, the resources that you add create a scope of the *city-networks* that you want Internet Monitor to monitor and that you want it to publish measurements for.

Resource added to monitor ("monitored resource")

A resource that you add to a monitor is a "monitored resource" in Internet Monitor. That is:

- Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or an AWS Fargate container.
- Each Network Load Balancer that you add in a Region is a monitored resource.
- Each WorkSpaces directory that you add in a Region is a monitored resource.

- Each CloudFront distribution that you add is a monitored resource.

Autonomous System Number (ASN)

In Internet Monitor, an ASN typically refers to an internet service provider (ISP), such as Verizon or Comcast. An ASN is a network provider that a client uses to access your internet application. An Autonomous System (AS) is a set of internet routable internet protocol (IP) prefixes that belong to a network or a collection of networks that are all managed, controlled, and supervised by one organization.

City-network (location and ASN)

A city-network is the location (such as a city) that clients access your application resources from and the ASN, typically an internet service provider (ISP), that clients access the resources through. To help control your bill, you can set a limit for the maximum number of city-networks for Internet Monitor to monitor for each monitor. You pay only for the actual number of city-networks that you monitor, up to the maximum number. For more information, see Choosing a city-network maximum limit.

Internet measurements

Internet Monitor also publishes internet measurements to log files in CloudWatch Logs every five minutes for the top 500 city-networks for your monitored application traffic.

These measurements quantify the performance score, availability score, bytes transferred (bytes in and bytes out), and round-trip time for your application's city-networks. These are measurements for the city-networks specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories. Optionally, you can choose to publish internet measurements and events for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket.

Metrics

Internet Monitor generates aggregated metrics for CloudWatch metrics, for global traffic to your application and global traffic to each AWS Region. For more information, see View Internet Monitor metrics or set alarms in CloudWatch Metrics.

Health event

Internet Monitor creates a health event to alert you to a specific problem that affects your application. Internet Monitor detects internet issues, such as increased network latency, across the world. It then uses its historical internet measurements from across the AWS global

infrastructure footprint to calculate the impact of current issues on your application, and creates health events. Internet Monitor, by default, creates health events based on both overall impact and local impact thresholds. To learn more about health events, see When Internet Monitor creates and resolves health events.

The default health event threshold, for both performance scores and availability scores, is 95%. If you like, you can specify your own custom thresholds for when Internet Monitor creates health events. For more information about configuring thresholds, see Change health event thresholds.

Each health event includes information about the impacted city-networks. You can view health events in the CloudWatch console, or by using an AWS SDK or AWS CLI with Internet Monitor API actions. Internet Monitor also sends Amazon EventBridge notifications for health events. For more information, see When Internet Monitor creates and resolves health events.

Internet event

Internet Monitor displays information about recent global health events, called internet events, on an internet weather map that is available to all AWS customers. You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events, internet events are not specific to individual customers or their application traffic. For more information, see Global internet weather map in Internet Monitor.

Thresholds

Internet Monitor creates health events based on both overall thresholds and local thresholds. You can change the default thresholds and configure other options, such as turning off local thresholds. For more information about configuring thresholds, see Change health event thresholds.

Performance and availability scores

By analyzing the data that AWS collects, Internet Monitor can detect when the performance and availability for your application has dropped, compared to estimated baselines that Internet Monitor calculates. To make it easier to see those drops, Internet Monitor reports the information to you as scores. A performance score represents the estimated percentage of traffic that is **not** seeing a performance drop. Similarly, an availability score represents the estimated percentage of traffic that is **not** seeing a availability drop. For more information, see How AWS calculates performance and availability scores.

Bytes transferred and monitored bytes transferred

Bytes transferred is the total number of bytes of ingress and egress traffic between an application in AWS and the city-network (that is, the location and the ASN, typically the internet service provider) where clients access an application. Monitored bytes transferred is a similar metric, but includes only bytes for monitored traffic.

Round-trip time

Round-trip time (RTT) is how long it takes for a request from a client user to return a response to the user. When RTT is aggregated across client locations (cities or other geographies), the value is weighted by how much of your application traffic is driven by each client location.

How Internet Monitor works

This section provides information about how Internet Monitor works. This includes descriptions of how AWS collects the data that it uses to help detect connectivity issues across the internet, and how performance and availability scores are calculated.

Contents

- How Internet Monitor focuses on just your application traffic footprint
- How AWS measures connectivity issues and calculates measurements
- Geolocation accuracy in Internet Monitor
- When Internet Monitor creates and resolves health events
- Health event report timing
- How Internet Monitor works with IPv4 and IPv6 traffic
- How Internet Monitor selects the subset of city-networks to include
- How the global internet weather map is created (Frequently Asked Questions)

How Internet Monitor focuses on just your application traffic footprint

Internet Monitor focuses monitoring on just the subset of the internet that's accessed by the users of your AWS resources, instead of broadly monitoring your website from every Region in the world as other tools do. It’s also a cost effective solution, affordable for large and small companies.

Internet Monitor uses the same powerful probes and issue-detection algorithms that AWS takes advantage of internally and alerts you to connectivity issues that affect your application by creating health events in Internet Monitor. Internet Monitor then gives you access to the resulting performance and availability map, by overlaying the traffic profile that it creates from your active viewers, based on your application resources.

Using this information, Internet Monitor shows you just relevant events (that is, the events from places where you have active viewers), and just the impact those events have on your overall viewer volume. So, how much impact an event has, percentage-wise, is based on your total traffic worldwide.

Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each AWS Region and edge location.

In addition, Internet Monitor publishes internet measurements to CloudWatch Logs internet every five minutes for the top 500 city-networks that send traffic to each monitor, to support using CloudWatch tools and other methods with your data. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.

The benefits of Internet Monitor include the following:

- Using Internet Monitor doesn't place additional load or cost on your application that's hosted on AWS.
- You don't need to include performance measurement code in your client-side resources, or in your application.
- You can get visibility into performance and availability across the internet that your application is connected to, including "last mile" information.

Note that because Internet Monitor creates measurements based on your AWS resources, Internet Monitor only creates events that are specific to your application traffic. Global internet issues in general are not reported. In addition, when the service location is an AWS Region, the measurements and events emitted are designed to represent connectivity at a Regional level and don’t accurately represent connectivity between an end user location and an Availability Zone.

How AWS measures connectivity issues and calculates measurements

Internet Monitor uses internet connectivity data between different AWS Regions and Amazon CloudFront points of presence (POPs) to different client locations through Autonomous System Numbers (ASNs), typically internet service providers (ISPs). This is the connectivity data that is used internally by AWS operators, on a daily basis, to proactively detect connectivity issues across the global internet.

For every AWS Region, we know which portions of the internet communicate with the Region and do the following:

- We actively monitor those portions of the internet, with a rolling 30-day window.
- We use both network and higher-level protocol probes, including both inbound and outbound probing.

AWS has active and passive probes that measure the latency (performance) at the 90th percentile and reachability (availability) from every AWS Region and from the CloudFront service to the entire internet. Abnormal patterns in connectivity between a service and a customer location are monitored, and then reported as alerts to the customer.

See the following sections for details:

- Calculating availability and RTT
- Calculating performance and availability scores
- Calculating TTFB and RTT (latency)
- Regional and Availability Zone measurements and aggregation

Calculating availability and RTT

Round-trip time (RTT) is how long it takes for a request from the user to return a response to the user. When round-trip time is aggregated across end user locations, the value is weighted by the amount of your traffic that is driven by each end user location.

As an example, with two end user locations, one serving 90% of traffic with a 5 ms RTT, and the other serving 10% of traffic with a 10 ms RTT, the result is an aggregated RTT of 5.5 ms (which comes from 5 ms * 0.9 + 10 ms * 0.1).

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Calculating performance and availability scores

AWS has substantial historical data about internet performance and availability between AWS services and different city-networks (locations and ASNs). By applying statistical analysis to the data, Internet Monitor can detect when the performance and availability for your application has dropped, compared to an estimated baseline that it has calculated. To make it easier to see those drops, that information is reported to you in the form of health scores: a performance score and an availability score.

Health scores are calculated at different granularities. At the finest granularity, we compute the health score for a geographic region, such as a city or a metro area, and an ASN (a *city-network*). We also roll up the individual health scores to overall health score numbers for an application in a monitor. If you view performance or availability scores without filtering for any specific geography or service provider, Internet Monitor provides overall health scores.

Overall health scores span your whole application for the specified time period. When the performance or availability score for your application's city-network pairs across your application reaches or drops below the corresponding health event threshold for performance or availability Internet Monitor triggers a health event. By default, the threshold is 95% for both overall performance and availability. Internet Monitor also creates health events based on local thresholds—if the option is enabled, as it is by default—based on values that you configure. To learn more about configuring health event thresholds, see Change health event thresholds.

When you explore information in the monitor and log files to investigate issues and learn more, you can filter by specific cities (locations), networks (ASNs or internet service providers), or both. So, you can use filters to see health scores for different cities, ASNs, or city-network pairs, depending on the filters that you choose.

- An *availability score* represents the estimated percentage of traffic that is **not** seeing an availability drop. Internet Monitor estimates the percentage of traffic experiencing a drop from the total traffic seen and availability metrics measurements. For example, an availability score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing an availability drop for that pair.
- A *performance score* represents the percentage of traffic that is **not** seeing a performance drop. For example, a performance score of 99% for an end user and service location pair is equivalent to 1% of the traffic experiencing a performance drop for that pair.

Calculating TTFB and RTT (latency)

Time to first byte (TTFB) refers to the time between when a client makes a request and when it receives the first byte of information from the server. AWS calculations for TTFB measure the time elapsed from Amazon EC2 or Amazon CloudFront to the Internet Monitor measurement node (including the last mile of the node). That is, Internet Monitor measures time from the user to the Amazon EC2 Region for TTFB for EC2, and from the user to CloudFront for TTFB for CloudFront.

For round-trip time (RTT), Internet Monitor includes the time from the city-network (that is, the client location and ASN, typically an internet service provider), as mapped by the public IP address, to the AWS Region. This means that Internet Monitor does not have last mile visibility for users who access the internet from behind a gateway or VPN.

Note that there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, Network Load Balancers, and WorkSpaces directories do not include last-mile latency.

Internet Monitor includes average TTFB information in the **Traffic optimization suggestions** section of the **Traffic insights** tab on the CloudWatch dashboard, to help you evaluate options for different setups for your application that can improve performance.

Regional and Availability Zone measurements and aggregation

Although Internet Monitor aggregates measurements and shares impact at a Regional level, it calculates impact at an Availability Zone (AZ) level. This means that, if, for an event, only one AZ is impacted and most of your traffic flows through that AZ, you do see impact for your traffic. However, for the same event, if your application traffic does not flow through an impacted AZ, you do not see impact.

Note that this applies only to resources that aren't WorkSpaces directories. WorkSpaces directories are measured only on a Regional level.

Geolocation accuracy in Internet Monitor

For location information, Internet Monitor uses IP-geolocation data supplied by MaxMind. The accuracy of the location information in Internet Monitor measurements depends on the accuracy of MaxMind's data.

Be aware that `Metro` level measurements might not be accurate for locations outside of the United States.

When Internet Monitor creates and resolves health events

Internet Monitor creates and closes health events for the application traffic that you monitor based on the current thresholds that are set. Internet Monitor has a default threshold configuration, and you can also set your own configuration for thresholds. Internet Monitor determines the overall impact that connectivity issues are having on your application, and the impact on local areas where your application has clients, and creates health events when the thresholds are crossed.

Internet Monitor calculates the impact of connectivity issues on a client location based on the historical data about internet performance and availability for network traffic that's available to the service through AWS. It applies the information relevant to your application, based on the geographic locations for ASNs and services where clients use your application: the city-network pairs that are affected. The locations are determined from the resources that you add to your monitor. Then Internet Monitor uses statistical analysis to detect when performance and availability has dropped, affecting the client experience for your application.

The performance and availability scores that Internet Monitor calculates are represented as the percentage of traffic that is **not** seeing a drop. Impact is the opposite of this: it's a representation of how much an issue is problematic for a customer's end users. So if there is a global availability drop of 93%, for example, the corresponding impact would be 7%.

When the performance or availability score for your application's city-network pairs globally reaches or drops below the corresponding health event threshold for performance or availability, this triggers Internet Monitor to generate a health event. By default, the threshold is 95% for both performance and availability. The values to meet, or drop below, the threshold are cumulative, so it could mean several smaller events combine to meet the threshold percentage, or that a single event meets or falls below the threshold level.

As long as performance or availability scores that triggered the event are at or below the corresponding health event threshold percentage for overall impact, the health event stays active. When the score or combined scores that triggered the event rise above the threshold, Internet Monitor resolves the health event.

Internet Monitor also creates health events based on local thresholds and the percentage of overall traffic that an issue has an impact on. You can configure options for local thresholds, or turn off local thresholds altogether.

The default health event threshold, for both performance scores and availability scores, is 95%. If you like, you can specify your own custom thresholds for when Internet Monitor creates

health events. For more information about configuring thresholds, see Change health event thresholds.

Health event report timing

Internet Monitor uses an aggregator to gather all signals about internet issues, to create health events in monitors within minutes.

When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by AWS or an ASN. Health event analysis continues after an event is resolved. Internet Monitor can update events with new information for up to an hour.

How Internet Monitor works with IPv4 and IPv6 traffic

Internet Monitor measures health toward a network over only IPv4, and shows you health events, and availability and performance metrics, if you serve traffic to that network over any IP family (IPv4 or IPv6). If you serve traffic from a dual-stack resource, such as a dual-stack CloudFront distribution, Internet Monitor raises a health event and shows a drop in a performance score or availability score only if IPv4 traffic has the same issues for the resource as IPv6 traffic does.

Note that the Internet Monitor metrics for overall bytes in and bytes out accurately reflect all internet traffic (IPv4 and IPv6).

How Internet Monitor selects the subset of city-networks to include

When you set a maximum limit for the number of city-networks monitored by your monitor or choose a percentage of traffic to monitor, Internet Monitor chooses the city-networks to include (monitor) based on highest recent traffic volume.

For example, if you set a maximum city-networks limit of 100, Internet Monitor monitors (up to) 100 city-networks based on your application traffic during a recent one hour period. Specifically, Internet Monitor monitors the top 100 city-networks that have had the most traffic in the most recent one hour window *before* the latest one hour window.

To illustrate this, say that the current time is 2:30 PM. In this scenario, the traffic that you see in your monitor was captured between 1:00 PM and 2:00 PM. The traffic volume measurement that Internet Monitor uses to determine the top 100 city-networks was captured between 12:00 PM and 1:00 PM.

How the global internet weather map is created (Frequently Asked Questions)

The Internet Monitor internet weather map is available on the Internet Monitor console to all authenticated AWS customers. This section includes details about how the internet weather map is created and how to use it.

What is the Internet Monitor internet weather map?

The internet weather map provides a visual representation of internet issues across the world. It highlights impacted client locations, that is, cities plus ASN (typically internet service providers). The map shows a combination of availability and performance issues that have recently impacted clients' internet experience for top client locations and AWS services globally.

Where does data for the map come from?

The data is based on a combination of active and passive probing of the internet. To learn more about how Internet Monitor measures data you can read the section How AWS measures connectivity issues.

How often is the map updated?

The internet weather map is updated every 15 minutes.

Which networks are tracked for outages?

AWS tracks networks all around the world that represent important IP prefixes used by customers for making internet connections to AWS. We scope outages to client locations that are top talkers for volume of traffic sent to and received from the AWS network.

What determines whether an internet event is included on the map?

Here are some high level criteria that we use to determine whether an internet event is included on the internet weather map:

- AWS detects that there is an availability or performance event.
- If the event is short lived, for example, it lasts less than 5 minutes, we ignore it.
- Then, if the event is in a client location that is classified as a top talker, it's considered an outage.

What thresholds are used for the internet weather map?

Thresholds for determining outages are not static for the internet weather map. Internet Monitor determines what constitutes an event based on detecting a deviation from expected values. You can learn more about how this works by reviewing how Internet Monitor

determines when to create health events for monitors that you create with the service. When you create a monitor, Internet Monitor generates internet traffic health measurements that are specific to your own application traffic. Internet Monitor also alerts you to health events for issues that affect your application's internet traffic.

What can I do with this data?

The internet weather map provides a quick summary of key internet events that happened around the world in the last 24 hours. It helps you to get a sense of the internet monitoring experience, without needing to onboard your own internet traffic to Internet Monitor. To leverage the full potential of the internet monitoring capabilities of AWS and to personalize it for your applications and services hosted on AWS, you can create a monitor in Internet Monitor.

When you create a monitor, you enable Internet Monitor to identify the specific internet paths that affect your application clients, and you get access to features and capabilities that can help you improve your client experience. You'll also be proactively notified of new internet issues that specifically impact your application traffic and clients.

How can I get more details about events?

Click an outage on the map to see details that include when an event started and ended, the impacted city and ASN, and what type of issue it was (that is, a performance issue or an availability issue).

To get more detailed information about events, and to get custom measurements for your application traffic, create a monitor in Internet Monitor.

Internet Monitor example use cases

This section describes several specific examples of use cases for Internet Monitor, with links to blog posts with more details. These examples illustrate how you can use the capabilities of Internet Monitor to monitor your application health and improve latency to enhance your users' experience.

Set up alerts and decide on actions to take

You can use Internet Monitor to get insights about average internet performance metrics over time, and about health events by city-network (client location and ASN, typically an internet service provider). Using Internet Monitor, you can identify the events that are impacting end user experience for applications hosted on Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon WorkSpaces, or Amazon CloudFront.

After you create a monitor, you have several options for how to be alerted about Internet Monitor health events. These include notifications based on CloudWatch Alarms using event metrics or Amazon EventBridge rules to filter for health events. You can choose different options for notifications or actions based on alarms, including, for example, AWS SMS notifications or updates to a CloudWatch log group.

To see an example with detailed guidance, see the following blog post: Introducing Internet Monitor.

Identify latency issues and improve TTFB to improve multiplayer gameplay experience

Use Internet Monitor to help you to quickly identify where game players in global cloud gaming apps are experiencing latency issues globally, and provide insights into improving performance. By identifying where the most players currently have the slowest time to first byte (TTFB), you know how to improve latency to make your biggest player base happier.

Now, when you're ready to deploy the next EC2 server for your game, choose the AWS Region that Internet Monitor suggests will lower TTFB in the area with the high latency and large group of players.

For details about setting up and using Internet Monitor for this use case, see the following blog post: Using Internet Monitor for a Better Gaming Experience.

Identify potential performance and internet connection issues for users on Amazon WorkSpaces

Internet Monitor provides you with the IP prefixes and ASN (typically, the internet service provider or ISP) for your users, which can be helpful to diagnose performance and internet connection issues for users to their WorkSpaces. You can also use this data to view your fleet as a whole and monitor your WorkSpaces user connections.

For more information about how to use Internet Monitor for this use case, see the following blog post: Using Internet Monitor with Amazon WorkSpaces Personal.

Global internet weather map in Internet Monitor

Internet Monitor displays a global internet weather map that is available to all AWS customers. To view the map, in the Amazon CloudWatch console, navigate to **Network Monitoring**, and then choose **Internet monitors**.

The internet weather map highlights internet events ("outages") all over the world that affect AWS customers, with the specific cities and networks (ASNs, typically internet service providers) where there are issues with performance or availability. The map includes internet events from the past 24 hours.

You don't need to create a monitor in Internet Monitor to view the internet weather map. Unlike health events in Internet Monitor, internet events are not specific to individual customers or their application traffic.

On the internet weather map, you can choose an internet event to learn details about it. For an internet event, you can see the start time, end time (if the event is over), the current status (Active or Resolved), and the outage issue type (Availability or Performance). To learn more about how the internet weather map is created and what is included, see the global internet weather map FAQ.

To view and work with detailed information that is specific to your application traffic and client locations, you can create a monitor in Internet Monitor for your application. Then, you'll see performance and availability patterns and events, current and historical, as well as get health event alerts, tailored to just your application footprint and customers. The internet weather map gives you an overall view, while a specific monitor filters the information to just the measurements and details that are relevant to your application. With a monitor, you can also explore historical metrics and get recommendations for improving client experience for your application. To learn more, see Getting started with Internet Monitor using the console.

Internet Monitor cross-account observability

With Internet Monitor cross-account observability, you can monitor your applications that span multiple AWS accounts within a single AWS Region.

You can use Amazon CloudWatch Observability Access Manager to set up one or more of your AWS accounts as a monitoring account. You'll provide the monitoring account with the ability to view data in your source account by creating a *sink* in your monitoring account. A sink is a resource that represents an attachment point in a monitoring account. For Internet Monitor, the resource attachment point is a monitor. You use the sink to create a link from your source account to your monitoring account. For more information, see CloudWatch cross-account observability.

Required resources

For proper functionality of CloudWatch Application Insights cross-account observability, make sure that the following telemetry types are shared through the CloudWatch Observability Access Manager.

- Monitors in Internet Monitor
- Metrics in Amazon CloudWatch
- Log groups in Amazon CloudWatch Logs

Pricing for Internet Monitor

With Internet Monitor, there are no upfront costs or long-term commitments. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A *city-network* is the location where clients access your application resources from and the network (ASN, such as an internet service provider or ISP) that clients access the resources through. Note that you are also charged standard CloudWatch prices for logs and any additional metrics, dashboards, alarms, or insights that you create.

You choose a percentage of traffic to monitor when you create a monitor. To help control your bill, you can also set a limit for the maximum number of city-networks to monitor. You can update the percentage of traffic to monitor or the maximum city-networks limit at any time by editing your monitor. The first 100 city-networks (across all monitors per account) are included. After that, you only pay for the actual additional number of city-networks that you monitor, up to the maximum number.

You pay only the actual additional number of city-networks that you monitor, up to the maximum number, with no charge for the first 100 city-networks (across all monitors per account). A flat amount equivalent to the cost of 100 city-networks is deducted from your monthly bill.

For example, a large global company could choose to monitor 100% of its internet-facing traffic, and set a city-networks maximum of 50,000, for one monitor with one resource. Assuming the traffic reached 50,000 city-networks, that portion of its bill would be around 2,700 USD/month. For another company, in fewer geographic areas, with one monitor with one resource and 200 city-networks, this portion of the bill would be around 13 USD/month. For more information, see Choose a city-networks maximum limit.

You can try out different options with the pricing calculator. To explore pricing options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

For more information about Internet Monitor and CloudWatch pricing, see the Amazon CloudWatch pricing page.

Getting started with Internet Monitor using the console

To help you get started with Internet Monitor, this chapter provides the steps for creating and configuring a *monitor*. You create a monitor in Internet Monitor for your application by naming it, and then adding AWS resources that your application uses.

You create a monitor in Internet Monitor for your application by adding AWS resources that it uses, and then setting several configuration options. The resources that you add, Amazon Virtual Private Cloud VPCs, Network Load Balancers (NLBs), CloudFront distributions, or WorkSpaces directories, provide the information for Internet Monitor to map internet traffic information for your application. After you create your monitor, wait 15-30 minutes to generate the traffic profile specific to where your application is used.

Then, use the Internet Monitor dashboard, or other tools, to visualize and explore performance and availability about your client usage. These tools provide insights for you using your application traffic's measurements gathered for you by the monitor.

The steps here walk you through setting up your monitor by using the console. To see examples of using the AWS Command Line Interface with the Internet Monitor API actions, to create a monitor, view events, and so on, see Examples of using the CLI with Internet Monitor.

Tasks

- Step 1: Create a monitor
- Step 2: Configure the monitor
- Step 3: View metrics and explore history
- Step 4: Get suggestions to improve latency
- Step 5 (Optional): Delete the monitor

Step 1: Create a monitor

To create a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name you want to use for this monitor in Internet Monitor.

5. Choose **Add resources**, and then select the resources to set the monitoring boundaries for Internet Monitor to use for this monitor.

> **Note**
>
> Be aware of the following:
>
> - To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
> - You can add only one type of resource to a single monitor. For example, VPCs or CloudFront distributions or WorkSpaces directories, but not a combination of different types.

6. Leave the default percentage of traffic as 100%, or choose another percentage of your internet traffic to monitor.
7. Choose **Create monitor**.

Step 2: Configure the monitor

After you create a monitor, you can edit the monitor at any time, for example, to change the application traffic percentage, update the maximum city-networks limit or add or remove resources. To make updates in the Internet Monitor console, follow the procedure in this section. Note that you can't change the name of a monitor.

For more information about configuring a monitor, see Edit a monitor in Internet Monitor.

To configure a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Make the desired updates. For example, to change the percentage of traffic to monitor, under **Application traffic to monitor**, select or enter a percentage.
6. Choose **Update**.

Step 3: View metrics and explore history

Visualize data about your internet traffic, from an overview perspective or by drilling down into details.

To visualize data and get insights for application traffic using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose a monitor to work with.
4. Choose from the following tabs:

   - **Overview** — Review a general summary of your monitor and your application traffic performance.
   - **Health events** — View current and historical health events that currently impact, or previously impacted, locations where clients access your application.
   - **Analyze** — See information about top monitored traffic in client locations (by traffic volume), summarized in several customizable ways. Visualize metrics and historical trends for health scores and metrics.

In the next section, learn about how Internet Monitor provides suggestions for improving latency for your application traffic.

Step 4: Get suggestions to improve latency

Get suggestions for how to optimize latency, so that your clients experience the best internet performance for your application.

Internet Monitor evaluates your monitored application traffic, and then makes suggestion about whether you can reduce latency, for example, by changing the AWS Regions that you've configured for your application.

For more information, see Get suggestions to optimize application performance in Internet Monitor (Optimize page).

To get suggestions for improving application latency using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose a monitor to work with.
4. Choose **Optimize**, and then view the top suggestions.

Step 5 (Optional): Delete the monitor

If you created a monitor as a test or if you're no longer using a monitor, you can delete it. Before you can delete a monitor, you must disable it.

To delete a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Disable**.
5. Choose the **Action** menu again, and then choose **Delete**.
6. Follow the guidance in the modal dialog to confirm deleting the monitor.

Configure a monitor using the console

This chapter includes procedures and recommendations for creating and configuring monitors in Internet Monitor.

The steps provided in these sections primarily use the AWS Management Console. You can also use Internet Monitor API operations with the AWS Command Line Interface (AWS CLI) or AWS SDKs to create and configure a monitor. For detailed information about working with Internet Monitor API operations, see the following resources:

- If you plan to work with Internet Monitor with the CLI, see Examples of using the CLI with Internet Monitor.
- For detailed information about working with Internet Monitor API operations, see the Internet Monitor API Reference.

Contents

- Create a monitor in Internet Monitor using the console

- Add resources to your monitor
- Choose a percentage of traffic to monitor for your application
- Use a monitor in Internet Monitor
- Edit a monitor in Internet Monitor
- Delete a monitor in Internet Monitor
- Advanced configuration options for a monitor

Create a monitor in Internet Monitor using the console

You create a monitor in Internet Monitor to visualize and explore performance and availability data about your application's client traffic. You create a monitor by adding AWS resources that your application uses, and then setting several configuration options. The resources that you add to the monitor provide the information — for example, through resource flow logs — for Internet Monitor to learn which internet traffic is specific to your AWS application.

After you create your monitor, wait 15 to 30 minutes before reviewing the monitor dashboard. Internet Monitor needs a few minutes to generate a traffic profile for where your application is used by your end users, and then to begin publishing data for your traffic.

Typically, it's simplest to create one monitor in Internet Monitor for one application. Within the same monitor, you can search through and sort measurements and metrics by different locations and ASNs, or other information. You don't need to create separate monitors for applications in different areas.

The steps provided here walk you through setting up your monitor by using the console. To work with Internet Monitor API actions using the AWS Command Line Interface, see Examples of using the CLI with Internet Monitor.

To create a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose **Create monitor**.
4. For **Monitor name**, enter the name that you want to use for this monitor.
5. Choose **Add resources**, and then select the resources that will determine the internet traffic profile for this monitor.

> **Note**
>
> Be aware of the following:
>
> - To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
> - You can specify only one type of resource for each monitor.

6. Choose a percentage of your application's internet traffic to monitor.
7. Optionally, under **Advanced settings**, specify one or more of the following additional options.

   - **City-networks maximum** — The default city-networks maximum value is `500000`. If you like, you can lower this limit, to restrict the number of city-networks (locations and ASNs) that Internet Monitor will monitor traffic for. You can change the city-networks maximum at any time by editing your monitor. For more information, see Choose a city-networks maximum limit.
   - **Amazon S3 bucket storage** — You can specify an Amazon S3 bucket name and custom prefix to publish internet measurements for your application's internet traffic to Amazon S3, for all monitored city-networks.

     Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each AWS Region and edge location. In addition, Internet Monitor publishes internet measurements for your application traffic to CloudWatch Logs every five minutes, to support using CloudWatch tools and other methods with your data. If you choose to publish measurements to S3, measurements are still published to CloudWatch Logs. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.
   - **Tags** — Add one or more tags for your monitor.
8. Choose **Create monitor**.

After you create a monitor, wait about 15-30 minutes for Internet Monitor to create a traffic profile and begin publishing your data. Then, you can view information about your application's internet traffic performance by navigating to the monitor dashboard in the console.

To view the Internet Monitor dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Network Monitoring**, then **Internet monitors**.
3. To see more information about a specific monitor, on the **Monitors** tab, choose a monitor.

Add resources to your monitor

When you create a monitor, you must associate your application's resources with it: Amazon Virtual Private Clouds (VPCs), Network Load Balancers, Amazon CloudFront distributions, Network Load Balancers (NLBs), or Amazon WorkSpaces directories. Then, Internet Monitor knows where your application's internet-facing traffic and clients are located, and it can create and maintain a traffic profile that determines the relevant measurements to publish for your monitor.

You can add the following types of resources to a monitor in Internet Monitor as *monitored resources*.

- **VPCs:** Each VPC that you add in a Region is a monitored resource. When you add a VPC, Internet Monitor monitors the traffic for any internet-facing application in the VPC, for example, an application hosted on an Amazon EC2 instance, behind a Network Load Balancer, or in an AWS Fargate container.
- **Network Load Balancers:** Each NLB that you add is a monitored resource.
- **CloudFront distributions:** Each CloudFront distribution that you add is a monitored resource.
- **WorkSpaces directories:** Each WorkSpaces directory that you add in a Region is a monitored resource.

When you monitor traffic for VPCs, traffic for applications that are hosted on load balancers behind the VPC is monitored. You can choose to monitor traffic for individual Network Load Balancer load balancers instead of monitoring a VPC with multiple load balancers. This can be helpful, for example, if you need to understand and configure features for better performance or efficiencies at the load balancer level. Or, you might need compliance information at the Network Load Balancer level.

When you add resources to a monitor in Internet Monitor, be aware of the following:

- Internet Monitor doesn't support adding different types of resources together in one monitor.

- To generate meaningful output with Internet Monitor, VPCs that you add must be connected to the internet by having an Internet Gateway configured.
- Internet Monitor doesn't support adding different types of resources together in one monitor.
- There are Regional differences for opt-in Regions to keep in mind when you add VPCs or NLBs as resources. For more information, see Supported AWS Regions for Internet Monitor.
- In addition, there are differences for resources about measuring last-mile latency. For Internet Monitor latency measurements, VPCs, NLBs, and WorkSpaces directories do not include last-mile latency.

Choose a percentage of traffic to monitor for your application

The coverage that you choose for the percentage of application traffic to monitor determines how many city-networks (client locations and ASNs, typically internet service providers) for your application are monitored, up to an optional city-networks maximum limit that you can also set.

You can choose the percentage of traffic to monitor when you create a monitor, or, with an existing monitor, by choosing **Edit monitor** on any Internet Monitor dashboard page in the console.

If you choose to monitor less than 100% of your application traffic, you might have an observability gap in with your monitor. That's because if there are health events that Internet Monitor creates where you aren't monitoring traffic, you won't be aware of those issues. With a traffic percentage set to less than 100%, you might also have less coverage for the performance and availability score information about client access to your application.

The following sections describe options to explore traffic percentage settings and coverage, and to get an idea about the impact of increasing or decreasing coverage.

- Explore changing your application traffic percentage
- View the number of city-networks monitored at different traffic percentage settings

Explore changing your application traffic percentage

You can explore values that you might want to change your application traffic percentage to, by viewing the number of city-networks monitored when you change the percentage. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can try increasing or decreasing the application traffic percentage for your monitor, and view the estimated number of your city-networks that would

be covered as a result. With this option, you can quickly see how changing your traffic percentage affects the number of city-monitors are monitored. This can help you to get a feel for what a good application traffic percentage to choose might be, for your application.

To explore monitoring coverage and update percentage of traffic monitored

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. In your list of monitors, choose a monitor.
4. On the **Configure** tab, in the **View and evaluate traffic coverage** section, you can evaluate the impact on the total number of city-networks that are monitored, depending on a traffic percentage that you choose. You can also update the percentage of traffic that you monitor or change the city-networks limit for your monitor.

   - **Explore traffic percentage options:** Under **Compare options for traffic coverage**, in the drop-down menu, choose one or more traffic percentages to graph and compare. For each traffic percentage that you choose, you can see the number of city-networks that will be monitored when you set that traffic percentage coverage.

     To learn more, see View number of city-networks monitored at different percentages.
   - **Change monitoring coverage:** Under **Explore other traffic coverage options**, choose **Update monitoring coverage**.

     In the **Explore and set traffic monitoring coverage** dialog, click the arrows to increase or decrease the percentage of traffic to monitor. By choosing 100% traffic, you can see how many city-networks are monitored with full coverage for monitoring your application.

     Note: To learn more about how the number of city-networks monitored (estimated here) might affect your costs, choose the link to the CloudWatch Pricing calculator, and then scroll down to Internet Monitor.

     To set a new percentage of traffic to monitor, choose **Update monitor coverage**. Or, to keep the current coverage level, choose **Cancel**.

View the number of city-networks monitored at different traffic percentage settings

You can view the number of city-networks that would be monitored for your application at different application traffic percentages. The procedure in this section provides step-by-step information.

In the Internet Monitor console, you can view graphs that show how coverage for your city-networks would change at different of application traffic percentages, over a time interval that you specify. This is a quick way to visualize and compare the monitoring coverage for your application at specific traffic percentages, all on one graph.

To view graphs of application traffic percentage and corresponding city-networks coverage

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. In your list of monitors, choose a monitor.
4. Choose the **Configure** page, and scroll down to **Traffic coverage**.
5. Under **Compare options for traffic coverage**, in the drop-down list, select one or more percentages. You can choose one or more application traffic percentages, and the graph of **Total monitored city-networks** is updated to display the monitoring coverage Internet Monitor provides for that traffic percentage. By choosing **City-networks at 100% traffic**, you can see how many city-networks are monitored with full coverage for monitoring your application.

Keep in mind the following:

- Traffic coverage is computed based on the number of city-networks in the previous hour of your application traffic. This means that, after you choose a specific percentage of traffic to monitor, fewer city-networks might be monitored for your application than is shown here in a traffic coverage comparison graph.
- To make sure that all your application traffic is monitored, set `TrafficPercentageToMonitor` to 100 and don't set `MaxCityNetworksToMonitor`. Alternatively, you can set `MaxCityNetworksToMonitor` to 500,000, the upper limit in Internet Monitor.
- If you set a city-networks maximum limit, the total number of monitored city-networks never exceeds that limit, regardless of the application traffic percentage option that you select.
- You can learn more about how the number of city-networks monitored might affect your costs. On the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

To set a new percentage of traffic to monitor, under **Explore other traffic coverage options**, choose **Update monitoring coverage**. In the dialog, choose a percentage of traffic, and then choose **Update monitor coverage**.

Use a monitor in Internet Monitor

There are several ways to use an Internet Monitor monitor after you create it: for example, you can view information in the CloudWatch dashboard, get information by using the AWS Command Line Interface, and set health alerts.

Your monitor provides information about your application and configuration preferences so that Internet Monitor can customize measurements and metrics to publish in events for you. Internet Monitor collects measurements from the global infrastructure footprint for AWS. These measurements are a tremendous amount of network performance and availability information, from all over the world. By using information from the resources that you add for your application, Internet Monitor publishes performance and availability measurements for you that is scoped to the city-networks (that is, client locations and ASNs, typically internet service providers or ISPs) where your application is active. So, the measurements and metrics in the Internet Monitor dashboard and in CloudWatch Logs —about availability, performance, monitored bytes transferred, and round-trip time—are specific to your client locations and ASNs.

Internet Monitor also determines when there are anomalies in performance and availability. By default, Internet Monitor overlays your traffic with the availability and performance measurements that AWS has collected for each source-destination pair in your client locations, to determine when there are notable drops in performance or availability. When there's significant degradation for your application's locations and scope, Internet Monitor generates a *health event*, and publishes information about the issue to your monitor.

After you create a monitor, you can use it to access or be alerted to the information that Internet Monitor provides, in the following ways:

- **Use the CloudWatch dashboard** to view and explore performance, availability, and health events; explore your application's historical data; and get insights into new ways to configure your application for better performance. To learn more, see the following:
  - Track real-time performance and availability in Internet Monitor (Overview page)
  - Analyze historical data in Internet Monitor (Analyze page)
  - Get suggestions to optimize application performance in Internet Monitor (Optimize page)

- **Configure health event thresholds** to change what triggers Internet Monitor to create a health event for your application. You can configure overall thresholds and local (city-network) thresholds. To learn more, see Change health event thresholds.
- **Use AWS CLI commands** with Internet Monitor API actions to view traffic profile information, view measurements, list health events, and so on. To learn more, see Examples of using the CLI with Internet Monitor.
- **Use standard CloudWatch tools,** such as CloudWatch Contributor Insights, CloudWatch Metrics explorer, and CloudWatch Logs Insights to visualize the data in CloudWatch. To learn more, see Exploring your data with CloudWatch tools and the Internet Monitor query interface.
- **Use Athena with S3 logs** to access and analyze Internet Monitor internet measurements for your application, if you turned on publishing measurements to S3.
- **Create Amazon EventBridge notifications** to alert you when Internet Monitor determines there is a health event. To learn more, see Using Internet Monitor with Amazon EventBridge.
- **Receive an AWS Health Dashboard notification** automatically, when Internet Monitor determines that an issue is caused by the AWS network. The notification includes the steps that AWS is taking to mitigate the problem.

Edit a monitor in Internet Monitor

Using the **Action** menu, you can edit a monitor in Amazon CloudWatch Internet Monitor after you create it. For example, you can edit a monitor to do the following:

- Change the percentage of application traffic to monitor
- Set or update the city-networks maximum limit
- Change health event thresholds for availability or performance scores
- Add or remove resources
- Enable or update publishing events to Amazon S3

Note that you can't change the name of a monitor after you create it.

To make changes to a monitor, use the following procedure.

To edit a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Make the desired updates. For example, to change the percentage of traffic to monitor, under **Application traffic to monitor**, select or enter a percentage.
6. Choose **Update**.

For more information about the options that you can update, see the following:

- To learn more about resources that you add in Internet Monitor, see Add resources to your monitor.
- To learn more about the application traffic percentage, see Choose a percentage of traffic to monitor for your application.
- To learn more about changing the threshold for health events, see Change health event thresholds.
- To learn more about the city-networks maximum limit, see Choose a city-networks maximum limit.
- To learn more about opting to publish events to S3, see Publish internet measurements to Amazon S3 in Internet Monitor.

Delete a monitor in Internet Monitor

Using the **Action** menu, you can delete a monitor in Amazon CloudWatch Internet Monitor. You first disable the monitor, and then delete it.

To delete a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network monitoring**, choose **Internet Monitor**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Disable**.
5. Choose the **Action** menu again, and then choose **Delete**.

Advanced configuration options for a monitor

This section provides the steps for configuring advanced options for a monitor in Internet Monitor. These configuration options are optional but can be useful in some scenarios.

For example, you might choose to set a city-network maximum limit if traffic for the application that you monitor with Internet Monitor occasionally spikes, and you want to help make sure that your bill for Internet Monitor is predictable.

Or, you might want to set custom or local thresholds for health events, because you want to pay close attention to issues in specific geographies where you have a concentration of clients.

The topics in this section provide detailed descriptions of each feature, and list the steps to configure options for your needs.

Contents

- Choose a city-networks maximum limit
- Change health event thresholds for a monitor
- Publish internet measurements to Amazon S3 in Internet Monitor

Choose a city-networks maximum limit

In addition to setting a traffic percentage for your monitor in Internet Monitor, you can also set a maximum limit for the number of city-networks monitored. This section describes how the city-networks limit can help you manage billing costs, and provides information and an example to help you determine a limit to set, if you choose to set one.

Internet Monitor can monitor traffic for some or all of the locations where clients access your AWS application resources. You can set a monitoring limit for the number of *city-networks*, that is, the client locations and the ASNs (typically internet service providers) that clients access your application through.

You choose a percentage of application traffic to monitor when you create your monitor. The default percentage is 100%. You can update the percentage at any time, by editing the monitor.

The maximum limit that you set for the number of city-networks helps to make sure that your bill is predictable. For more information, see Amazon CloudWatch Pricing. You can also learn how different values for the number of city-networks actually monitored can affect your bill by using

the CloudWatch price calculator. To explore options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor.

To update your monitor and change the maximum city-networks limit, see Edit a monitor in Internet Monitor.

How billing works with city-networks maximum limits

Setting a maximum limit for the number of city-networks monitored can help prevent unexpected costs in your bill. This is useful, for example, if your traffic patterns vary widely. Billing costs increase for each city-network that is monitored after the first 100 city-networks, which are included (across all monitors per account). If you set a city-networks maximum limit, it caps the number of city-networks that Internet Monitor monitors for your application, regardless of the percentage of traffic that you choose to monitor.

You only pay for the number of city-networks that are actually monitored. The city-network maximum limit that you choose lets you set a cap on the total that can be included when Internet Monitor monitors traffic with your monitor. You can change the maximum limit at any time by editing your monitor.

To explore options, on the Pricing calculator for CloudWatch page, scroll down to Internet Monitor. For more information on Internet Monitor pricing, see the Internet Monitor section on the Amazon CloudWatch Pricing page.

How to choose a city-networks maximum limit

Optionally, you can set a city-networks maximum limit. To help you decide on a maximum limit that you might want to select, consider how much traffic you want to monitor for your application. Be aware that if you choose 100% for the *traffic percentage to monitor* for your monitor, and then specify a city-networks maximum limit, depending on the limit that you choose, you might not monitor 100% of your application traffic. The city-networks maximum that you set takes precedence over the traffic percentage to monitor that you set.

To view how the percentage of traffic to monitor that you choose affects the number of city-monitors that are included for your application monitoring, which can help you decide whether to set a city-networks maximum limit, follow the steps in View the number of city-networks monitored at different traffic percentage settings.

To explore your options in more detail, you can use Internet Monitor metrics, as described in the following examples. These examples show how to select a maximum city-networks limit that is best for you, depending on the breadth of application internet traffic coverage you want. Using the

queries for Internet Monitor metrics in CloudWatch Metrics can help you understand more about your application internet traffic coverage.

Example of determining a city-networks maximum limit

As an example, say that you've set a monitoring maximum limit of 100 city-networks and that your application is accessed by clients across 2637 city-networks. In CloudWatch Metrics, you'd see the following Internet Monitor metrics returned:

```
CityNetworksMonitored 100
TrafficMonitoredPercent  12.5
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

From this example, you can see that you're currently monitoring 12.5% of your internet traffic, with the maximum limit set to 100 city-networks. If you want to monitor 90% of your traffic, the next metric provides information about that: `CityNetworksFor90PercentTraffic` indicates that you would need to monitor 2,143 city-networks for 90% coverage. To do that, you would update your monitor and set the maximum city-networks limit to 2,143.

Similarly, say you'd like to have 100% internet traffic monitoring for your application. The next metric, `CityNetworksFor100PercentTraffic`, indicates that to do this, you should update your monitor to set the maximum city-networks limit to 2,637.

If you now set the maximum to 5,000 city-networks, since that's greater than 2,637, you see the following metrics returned:

```
CityNetworksMonitored 2637
TrafficMonitoredPercent  100
CityNetworksFor90PercentTraffic  2143
CityNetworksFor100PercentTraffic  2637
```

From these metrics, you can see that with the higher limit, you monitor all 2,637 city-networks, which is 100% of your internet traffic.

Change health event thresholds for a monitor

Internet Monitor uses a default threshold to determine when to create a health event for your monitor. Optionally, you can change that default global threshold, to set another value. You can also set local threshold. This section describes how global and local thresholds work together, and provides steps for setting custom thresholds.

You can change the overall threshold that triggers Internet Monitor to create a health event. The default health event threshold, for both performance scores and availability scores, is 95%. That is, when the overall performance or availability score for your application falls to 95% or below, Internet Monitor creates a health event. For the overall threshold, the health event can be triggered by a single large issue, or by the combination of multiple smaller issues.

You can also change the local—that is, city-network—threshold, combined with a percentage of the overall level of impact, that—in combination—will trigger a health event. By setting a threshold that creates a health event when a score drops below the threshold for one or more city-networks (locations and ASNs, typically ISPs), you can get insights into when there are issues in locations with lower traffic, for example.

An additional local threshold option works together with the local threshold for availability or performance scores. The second factor is the percentage of your overall traffic that must be impacted before Internet Monitor creates a health event based on the local threshold.

By configuring the threshold options for overall traffic and local traffic, you can fine-tune how frequently health events are created, to align with your application usage and your needs. Be aware that when you set the local threshold to be lower, typically more health events are created, depending on your application and the other threshold configuration values that you set.

In summary, you can configure health event thresholds—for performance scores, availability scores, or both—in the following ways:

- Choose different global thresholds for triggering a health event.
- Choose different local thresholds for triggering a health event. With this option, you can also change the percentage of impact on your overall application that must be exceeded before Internet Monitor creates an event.
- Choose to turn off triggering a health event based on local thresholds, or enable local threshold options.

To update health event thresholds for performance scores, availability scores, or both, follow these steps.

To change threshold configuration options

1. In the AWS Management Console, navigate to CloudWatch, and then, in the left navigation pane, choose Internet Monitor.

2. On the **Configure** page, in the **Health event thresholds** section, choose **Update thresholds**.
3. On the **Set health event threshold**page, choose the new values and options that you want for thresholds and other options that trigger Internet Monitor to create a health event. You can do any of the following:

   - Choose a new value for **Availability score threshold**, **Performance score threshold**, or both.

     The graphs in the sections for each setting display the current threshold setting and the actual recent health event scores, for availability or performance, for your application. By viewing the typical values, you can get an idea of values that you might want to change a threshold to.

     Tip: To view a larger graph and change the timeframe, choose the expander in the upper right corner of the graph.

   - Choose to turn on or off a local threshold for availability or performance, or both. When an option is enabled, you can set the threshold and impact level for when you want Internet Monitor to create a health event.
4. After you configure threshold options, save your updates by choosing **Update health event thresholds**.

To learn more about how health events work, see When Internet Monitor creates and resolves health events.

Publish internet measurements to Amazon S3 in Internet Monitor

You can choose to have Internet Monitor publish internet measurements to Amazon S3 for your internet-facing traffic to the monitored city-networks (client locations and ASNs, typically internet service providers) in your monitor, up to the 500,000 city-networks service limit. Internet Monitor automatically publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks for each monitor. Measurements that it publishes to S3 include the top 500 that are published to CloudWatch Logs.

You can choose the option to publish to S3, and specify the bucket to publish the measurements, to when you create or update your monitor. The bucket must already be created in S3 before you can specify it in Internet Monitor. There's a service limit of 500,000 city-networks for internet measurements published to S3. Internet Monitor publishes internet measurements to S3 as events, a series of compressed log file objects that are stored in the bucket.

When you create the S3 bucket for Internet Monitor to publish measurements to, make sure that you follow the permissions guidance provided by CloudWatch Logs. Doing so ensures that Internet Monitor can publish logs directly to S3, and that AWS can, if needed, create and change the resource policies associated with the log group receiving the logs. For more information, see Logs sent to CloudWatch Logs in the Amazon CloudWatch Logs User Guide.

The published log files are compressed. If you open the log files using the Amazon S3 console, they are decompressed and the internet measurement events are displayed. If you download the files, you must decompress them to view the events.

You can also query the internet measurements in the log files using Amazon Athena. Amazon Athena is an interactive query service that makes it easier to analyze data in Amazon S3, by using standard SQL. For more information, see Use Amazon Athena to query internet measurements in Amazon S3 log files.

Examples of using the CLI with Internet Monitor

This section includes examples for using the AWS Command Line Interface with Internet Monitor operations.

Before you begin, make sure that you log in to use the AWS CLI with the same AWS account that has the Amazon VPC VPCs, Network Load Balancers, Amazon CloudFront distributions, or Amazon WorkSpaces directories that you want to monitor. Internet Monitor doesn't support accessing resources across accounts. For more information about using the AWS CLI, see the AWS CLI Command Reference. For more information about using API actions with Internet Monitor, see the Internet Monitor API Reference Guide.

Topics

- Create a monitor
- View monitor details
- List health events
- View specific health event
- View monitor list
- Edit monitor
- Delete monitor

Create a monitor

When you create a monitor in Internet Monitor, you provide a name and associate resources with the monitor to show where your application's internet traffic is. You specify a traffic percentage that defines how much of your application traffic is monitored. That also determines the number of city-networks, that is, client locations and ASNs, typically internet service providers or ISPs, that are monitored. You can also opt to set a limit for the maximum number of city-networks to monitor for your application resources, to help control your bill. For more information, see Choose a city-networks maximum limit.

Finally, you can choose if you want to publish all internet measurements for your application to Amazon S3. Internet measurements for the top 500 city-networks (by traffic volume) are automatically published to CloudWatch Logs by Internet Monitor, but you can choose to publish all measurements to S3 as well.

To create a monitor with the AWS CLI, you use the `create-monitor` command. The following command creates a monitor that monitors 100% of traffic but sets a maximum city-networks limit of 10,000, adds a VPC resource, and opts to publish internet measurements to Amazon S3.

> **Note**
>
> Internet Monitor publishes to CloudWatch Logs internet measurements every five minutes for the top 500 city-networks (client locations and ASNs, typically internet service providers or ISPs) that send traffic to each monitor. Optionally, you can choose to publish internet measurements for all monitored city-networks (up to the 500,000 city-networks service limit) to an Amazon S3 bucket. For more information, see Publish internet measurements to Amazon S3 in Internet Monitor.

```
aws internetmonitor create-monitor --monitor-name "TestMonitor" \
    --traffic-percentage-to-monitor 100 \
    --max-city-networks-to-monitor 10000 \
    --resources "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --internet-measurements-log-delivery S3Config="{BucketName=amzn-s3-demo-
bucket,LogDeliveryStatus=ENABLED}"
```

```
{
    "Arn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
```

```
}
```

> **Note**
>
> You can't change the name of a monitor.

View monitor details

To view information about a monitor with the AWS CLI, you use the `get-monitor` command.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor"
```

```
{
    "ClientLocationType": "city",
    "CreatedAt": "2022-09-22T19:27:47Z",
    "ModifiedAt": "2022-09-22T19:28:30Z",
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "MonitorName": "TestMonitor",
    "ProcessingStatus": "OK",
    "ProcessingStatusInfo": "The monitor is actively processing data",
    "Resources": [
        "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889"
    ],
    "MaxCityNetworksToMonitor": 10000,
    "Status": "ACTIVE"
}
```

List health events

When performance degrades for your application's internet traffic, Internet Monitor creates health events in your monitor. To see a list of current health events with the AWS CLI, use the `list-health-events` command.

```
aws internetmonitor list-health-events --monitor-name "TestMonitor"
```

```
{
    "HealthEvents": [
        {
            "EventId": "2022-06-20T01-05-05Z/latency",
```

```
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:15:14Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.21,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 60.370000000000005,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                }
            ],
            "StartedAt": "2022-06-20T01:05:05Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-05-05Z/latency"
        },
        {
            "EventId": "2022-06-20T01-17-56Z/latency",
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:30:23Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.29,
            "ClientLocations": [
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 75.32,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
```

```
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 22.91,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:17:56Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-17-56Z/latency"
        },
        {
            "EventId": "2022-06-20T01-34-20Z/latency",
```

```
            "Status": "RESOLVED",
            "EndedAt": "2022-06-20T01:35:04Z",
            "ServiceLocations": [
                {
                    "Name": "us-east-1"
                }
            ],
            "PercentOfTotalTrafficImpacted": 1.15,
            "ClientLocations": [
                {
                    "City": "Lockport",
                    "PercentOfClientLocationImpacted": 39.45,
                    "PercentOfTotalTraffic": 2.01,
                    "Country": "United States",
                    "Longitude": -78.6913,
                    "AutonomousSystemNumber": 26101,
                    "Latitude": 43.1721,
                    "Subdivision": "New York",
                    "NetworkName": "YAHOO-BF1"
                },
                {
                    "City": "Toronto",
                    "PercentOfClientLocationImpacted": 29.770000000000003,
                    "PercentOfTotalTraffic": 1.05,
                    "Country": "Canada",
                    "Longitude": -79.3623,
                    "AutonomousSystemNumber": 14061,
                    "Latitude": 43.6547,
                    "Subdivision": "Ontario",
                    "CausedBy": {
                        "Status": "ACTIVE",
                        "Networks": [
                            {
                                "AutonomousSystemNumber": 16509,
                                "NetworkName": "Amazon.com"
                            }
                        ],
                        "NetworkEventType": "AWS"
                    },
                    "NetworkName": "DIGITALOCEAN-ASN"
                },
                {
                    "City": "Hangzhou",
                    "PercentOfClientLocationImpacted": 2.88,
```

```
                    "PercentOfTotalTraffic": 0.7799999999999999,
                    "Country": "China",
                    "Longitude": 120.1612,
                    "AutonomousSystemNumber": 37963,
                    "Latitude": 30.2994,
                    "Subdivision": "Zhejiang",
                    "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
                }
            ],
            "StartedAt": "2022-06-20T01:34:20Z",
            "ImpactType": "PERFORMANCE",
            "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
TestMonitor/health-event/2022-06-20T01-34-20Z/latency"
        }
    ]
}
```

View specific health event

To see a more detailed information about a specific health event with the CLI, run the `get-health-event` command with your monitor name and a health event ID.

```
aws internetmonitor get-monitor --monitor-name "TestMonitor" --event-id "health-event/
TestMonitor/2021-06-03T01:02:03Z/latency"
```

```
{
    "EventId": "2022-06-20T01-34-20Z/latency",
    "Status": "RESOLVED",
    "EndedAt": "2022-06-20T01:35:04Z",
    "ServiceLocations": [
        {
            "Name": "us-east-1"
        }
    ],
    "EventArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor/
health-event/2022-06-20T01-34-20Z/latency",
    "LastUpdatedAt": "2022-06-20T01:35:04Z",
    "ClientLocations": [
        {
            "City": "Lockport",
            "PercentOfClientLocationImpacted": 39.45,
            "PercentOfTotalTraffic": 2.01,
```

```
            "Country": "United States",
            "Longitude": -78.6913,
            "AutonomousSystemNumber": 26101,
            "Latitude": 43.1721,
            "Subdivision": "New York",
            "NetworkName": "YAHOO-BF1"
        },
        {
            "City": "Toronto",
            "PercentOfClientLocationImpacted": 29.770000000000003,
            "PercentOfTotalTraffic": 1.05,
            "Country": "Canada",
            "Longitude": -79.3623,
            "AutonomousSystemNumber": 14061,
            "Latitude": 43.6547,
            "Subdivision": "Ontario",
            "CausedBy": {
                "Status": "ACTIVE",
                "Networks": [
                    {
                        "AutonomousSystemNumber": 16509,
                        "NetworkName": "Amazon.com"
                    }
                ],
                "NetworkEventType": "AWS"
            },
            "NetworkName": "DIGITALOCEAN-ASN"
        },
        {
            "City": "Shenzhen",
            "PercentOfClientLocationImpacted": 4.07,
            "PercentOfTotalTraffic": 0.61,
            "Country": "China",
            "Longitude": 114.0683,
            "AutonomousSystemNumber": 37963,
            "Latitude": 22.5455,
            "Subdivision": "Guangdong",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        },
        {
            "City": "Hangzhou",
            "PercentOfClientLocationImpacted": 2.88,
            "PercentOfTotalTraffic": 0.7799999999999999,
            "Country": "China",
```

```
            "Longitude": 120.1612,
            "AutonomousSystemNumber": 37963,
            "Latitude": 30.2994,
            "Subdivision": "Zhejiang",
            "NetworkName": "Hangzhou Alibaba Advertising Co.,Ltd."
        }
    ],
    "StartedAt": "2022-06-20T01:34:20Z",
    "ImpactType": "PERFORMANCE",
    "PercentOfTotalTrafficImpacted": 1.15
}
```

View monitor list

To see a list of all monitors in your account with the CLI, run the `list-monitors` command.

```
aws internetmonitor list-monitors
```

```
{
    "Monitors": [
        {
            "MonitorName": "TestMonitor",
            "ProcessingStatus": "OK",
            "Status": "ACTIVE"
        }
    ],
    "NextToken": " zase12"
}
```

Edit monitor

To update information about your monitor by using the CLI, use the `update-monitor` command and specify the name of the monitor to update. For example, you can update the percentage of traffic to monitor, the limit of the maximum number of city-networks to monitor, add or remove the resources that Internet Monitor uses to monitor traffic, and change the monitor status from `ACTIVE` to `INACTIVE`, or vice versa. Note that you can't change the name of the monitor.

The response for an `update-monitor` call returns just the `MonitorArn` and the `Status`.

The following example shows how to use the `update-monitor` command to change the maximum number of city-networks to monitor to `50000`:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --max-city-networks-to-
monitor 50000
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": " ACTIVE "
}
```

The following example shows how to add and remove resources:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" \
    --resources-to-add "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-11223344556677889" \
    --resources-to-remove "arn:aws:ec2:us-east-1:111122223333:vpc/vpc-2222444455556666"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "ACTIVE"
}
```

The following example shows how to use the `update-monitor` command to change the monitor status to `INACTIVE`:

```
aws internetmonitor update-monitor --monitor-name "TestMonitor" --status "INACTIVE"
```

```
{
    "MonitorArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/TestMonitor",
    "Status": "INACTIVE"
}
```

Delete monitor

You can delete a monitor with the CLI by using the `delete-monitor` command. First, you must set the monitor to be inactive. To do that, use the `update-monitor` command to change the status to `INACTIVE`. Confirm that the monitor is inactive by using the `get-monitor` command and checking the status.

When the monitor status is `INACTIVE`, then you can use the CLI to run the `delete-monitor` command to delete the monitor. The response for a successful `delete-monitor` call is empty.

```
aws internetmonitor delete-monitor --monitor-name "TestMonitor"
```

```
{}
```

Monitor and optimize with the Internet Monitor dashboard

Using the Internet Monitor dashboard in the AWS Management Console, you can visualize data and get insights and suggestions about your AWS application's internet traffic, and configure options for your monitor.

After you create a monitor to monitor your application's internet performance and availability, Internet Monitor stores internet measurements for pairs of your client locations and ASNs, or *city-networks*. Internet Monitor also creates aggregated CloudWatch metrics for traffic to your application, and to each AWS Region and edge location. You can filter, explore, and get action-oriented suggestions from your monitor's information in several different ways. The Internet Monitor dashboard guides you through viewing and getting insights about the data for your monitored traffic.

To get started, on the CloudWatch console, under **Network Monitoring**, choose **Internet monitors**. Then, select a monitor to work with.

> **Note**
>
> This section primarily describes how to filter and view Internet Monitor metrics using the AWS Management Console. Alternatively, you can use Internet Monitor API operations with the AWS CLI or an SDK to work directly with Internet Monitor events stored in CloudWatch Logs files. For more information, see Using your monitor and measurements information. For more information about using API operations, see Examples of using the CLI with Internet Monitor and the Internet Monitor API Reference.

There are five pages (tabs) in the Internet Monitor dashboard:

- On the **Overview** page, you can get an overall view of your monitored traffic, including current performance and availability information, a summary of recent and current health events, and the top suggestion for potentially improving performance for your clients.
- On the **Health events** page, you can see current and historical health events that currently impact, or previously impacted, locations where clients access your application.

- On the **Analyze** page, you can view information about top monitored traffic in client locations (by traffic volume), summarized in several customizable ways. You can also see historical trends for health scores and metrics. You can filter by location, ASN, date, and so on, and visualize metrics for your internet traffic over time.
- On the **Optimize** page, Internet Monitor predicts your application's performance improvement for top AWS Regions (or Amazon CloudFront), based on your traffic patterns and past performance. For each top configuration, associated tables provide a breakdown of reduced latency by client location. On a second page, you can select multiple Regions (and, if you like, include CloudFront configurations) to compare latency reductions. For each configuration (Region) that you selected, the page displays an associated table of latency details, listed by city location.
- On the **Configure** page, you can see monitor details and configure options, such as the percentage of traffic to monitor.

In addition to these dashboard options, you can use tools for deeper dives into details of the metrics that Internet Monitor collects with your monitor. Internet Monitor generates and publishes log files with the measurements about your traffic, so you can use other CloudWatch tools in the console to further visualize the data published by Internet Monitor, including CloudWatch Contributor Insights, CloudWatch Metrics, and CloudWatch Logs Insights. For more information, see Exploring your data with CloudWatch tools and the Internet Monitor query interface.

Learn about using Internet Monitor to explore your performance and availability measurements in the following sections.

Topics

- Track real-time performance and availability in Internet Monitor (Overview page)
- View health events and metrics in Internet Monitor (Health events page)
- Analyze historical data in Internet Monitor (Analyze page)
- Get suggestions to optimize application performance in Internet Monitor (Optimize page)
- Monitoring details in Internet Monitor (Configure page)

Track real-time performance and availability in Internet Monitor (Overview page)

The **Overview** page in the Internet Monitor console shows you a high-level view of performance and availability for the traffic that your monitor tracks, and a timeline of when health events have

impacted your monitored traffic. The page also provides the top suggestion for a configuration change that could reduce latency for clients who use your application in your top client location (by traffic volume).

Traffic overview and Status

The **Traffic overview** section provides an overall look at your application's availability and performance. Note that this section shows *aggregated* overall performance and availability scores that consider all of the traffic for applications towards all end users and service locations. You can see health scores for specific client locations and service locations by searching and filtering measurements information on the **Analyze** tab.

Under **Status**, you can see if your monitor is actively creating data for your monitor or is waiting for data to be available. You can also see the percentage of your application's traffic that you're monitoring. If you want to change the percentage, see the **Configure** page.

Internet Monitor uses a statistical process to create availability and performance scores for your monitored traffic. AWS has substantial historical data about internet performance and availability for network traffic between geographic locations for different ASNs and AWS services. Internet Monitor uses the connectivity data that AWS has captured from its global networking footprint to calculate a baseline of availability and performance for internet traffic. This is the same data that we use at AWS to monitor our own internet uptime and availability.

With those measurements as a baseline, Internet Monitor can detect when the performance and availability for your application has dropped, compared to the baseline. To make it easier to see those drops, we report that information to you as a performance score and an availability score.

For more information, see How AWS calculates performance and availability scores.

Health events timeline

The **Health events timeline** graph displays health events that have occurred during the past 24 hours. A summary below the graph shows your application's current and recent impact. For details, you can choose **See more health events**.

To change the thresholds for health events, go to the **Configure** page.

Reduce latency for your top Region

Internet Monitor automatically evaluates the AWS Region that your current application configuration uses most (that is, the Region with the highest client volume), and determines if another Region could provide a better aggregate time to first byte (TTFB) for your clients.

Note that because this is the aggregate TTFB, if you move traffic from one Region to another, TTFB for most locations is expected to improve but clients in some Regions could see no change or reduced performance..

To explore more latency improvement suggestions, including details at more granular levels (such as by client location), see the **Optimize** page.

View health events and metrics in Internet Monitor (Health events page)

The **Health events** page in the Internet Monitor console provides a map of health events that impact the client locations and ASNs for your application. You can click circles on the map for more details about an event. The **Health events** tables lists locations that have been impacted by an event, and specifics about the impact.

Internet traffic overview

The **Internet traffic overview** map shows you the internet traffic and health events that are specific to the locations and ASNs that your clients access your application from. The countries that are gray on the map are those that include traffic for your application.

Each circle on the map indicates a health event in an area, for a time period that you select. Internet Monitor creates health events when it detects a problem, at a specific (but customizable) threshold, with connectivity between one of your resources hosted in AWS and a city-network where a client is accessing your application.

Choose a circle on the map to display more details about the health event for that location. In addition, for clusters that have health events, you can see detailed information in the **Health events** table below the map.

Note that Internet Monitor creates health events in a monitor when it determines that an event has significant impact on your application. The map is blank if there aren't any health events that exceed the threshold for impact on traffic for your client locations in the time period that you've selected. For more information, see When Internet Monitor creates and resolves health events.

Health events

The **Health events** table lists client locations that have been affected by health events, along with information about the events. The following columns are included in the table.

Column	Description
Event type	Specifies whether current events are *overall* health events or *local* health events, or if the health event is in the *past*.
Client location	The location of the end users who were impacted by the event, who experienced increased latency or reduced availability. To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.
ISP name (ASN)	The network that the traffic traveled over. Typically, this is the internet service provider (ISP) or Autonomous System Number (ASN) for the network traffic.
Service location	The AWS location for the network traffic, which can be an AWS Region or an internet edge location.
Traffic impact	How much impact was caused by the event, in increased latency or reduced availability. For latency, this is the percentage of how much latency increased during the event compared to typical performance for traffic, from this client location to this AWS location using this client network.
Impact type	The type of impact for the health event. Health events are typically caused by latency increases (performance issues) or reachability (availability issues).

Column	Description
	You might also be able to click on the impact type to see the cause of the impairment. When possible, Internet Monitor analyzes the origin of a health event, to determine whether it was caused by AWS or an ASN (internet service provider). Note that this analysis continues after the event is resolved. Internet Monitor can update events with new information for up to an hour.

If you choose one of the client locations in the **Health events** table, you can see more details about the health event at that location. For example, you can see when the event started, when it ended, and the local traffic impact.

Network path visualization

If Internet Monitor has finished impairment analysis for an event, you can view **Network path visualization** to see the full network path for traffic to a client location. The full path shows you each node along the network path for your application for the health event, between the AWS location and the client, for a client-location pair.

When Internet Monitor has determined the cause of an impairment, Internet Monitor adds a dashed red circle around the node. Impairments can be caused by ASNs, typically internet service providers (ISPs), or the cause can be AWS. If there were multiple causes for an impairment, multiple nodes are circled.

Analyze historical data in Internet Monitor (Analyze page)

On the **Analyze** page in the Internet Monitor console, you can view your application's the top client locations for the traffic that you monitor, by traffic volume. You can also view graphs showing performance and availability scores for your traffic over time, as well as graphs of other internet traffic metrics for your application's monitored traffic.

To start exploring Internet Monitor data for your application traffic, select a time period. Then, choose a specific geographical location, such a city, and (optionally) other filters. Internet Monitor

applies the filters to your data, and then you can see graphs of the data that show measurements for your application. The graphs included on the **Analyze** page include your application's performance score, availability score, monitored bytes transferred (for VPCs, Network Load Balancers, and CloudFront distributions) or client connection counts (for WorkSpaces directories), and round-trip time (RTT) for your application over time.

The options at the top of the **Analyze** page determine the timeframe and types of traffic shown in the graphs on the page. You can filter by client locations or ASN, or choose to show traffic graphs at a specific granularity (the default is city level).

Top client locations

The **Top client locations** graph displays your top monitored traffic locations, by default. You can choose another field to sort the graph by, or you can sort the graph in other ways, for example, by lowest traffic locations.

The filters that you choose for the page determine the Regions, timeframe, and so on for the locations.

Traffic health scores

This section shows you graphs of traffic health scores and metrics for your monitored traffic. These graphs reflect data for the filters that you choose at the top of the page.

The **Traffic health scores** graph shows you performance and availability information for your local and overall traffic by calling out health events that have impacted your monitored client traffic. AWS has substantial historical data about internet performance and availability for network traffic between geographic locations for different ASNs and AWS services. Internet Monitor uses the connectivity data that AWS has captured from its global networking footprint to calculate a baseline of performance and availability for internet traffic. This is the same data that we use at AWS to monitor our own internet uptime and availability.

With those measurements as a baseline, Internet Monitor can detect when the performance and availability for your application has dropped, compared to the baseline. To make it easier to see those drops, we report that information to you as a performance score and an availability score. For more information, see How AWS calculates performance and availability scores.

Additional graphs show the monitored bytes transferred (for VPCs, Network Load Balancers, and CloudFront distributions) or client connection counts (for WorkSpaces directories), and round-trip time (RTT) for your application traffic.

Note that when round-trip time (RTT) is aggregated across end-user locations, the value is weighted by the amount of your traffic that is driven by each client location. For example, with two client locations, one serving 90% of traffic with a 5 ms RTT, and the other serving 10% of traffic with a 10 ms RTT, the result is an aggregated RTT of 5.5 ms (which comes from 5 ms * 0.9 + 10 ms * 0.1).

You can also explore the internet measurements that Internet Monitor stores for your monitored traffic by using CloudWatch tools or other methods. For more information, see Exploring your data with CloudWatch tools and the Internet Monitor query interface. In addition, you can create CloudWatch alarms based on Internet Monitor data, for example, to notify you of health events. For more information, see Create alarms with Internet Monitor.

Get suggestions to optimize application performance in Internet Monitor (Optimize page)

Use the **Optimize** page in the Internet Monitor console to get suggestions for how to optimize application performance for your clients. Internet Monitor evaluates your monitored application traffic, and determines if you can reduce latency by changing the AWS Regions that you've configured for your application. Optionally, you can also view latency changes if you choose to include Amazon CloudFront in the suggestions.

You can review suggestions for your application's top Regions by traffic volume, or for top client locations, also by traffic volume.

Suggestions to reduce latency for top Regions

To help you quickly understand your best options for reducing latency for your clients, Internet Monitor automatically provides suggestions for improving latency in your application for your top Regions (by traffic volume).

You can also explore configuration changes for all the Regions where your application serves clients. This includes getting details about each suggested change at deeper levels of granularity, for example, by specific client location. To explore all Regional configurations and expected latency changes for your application, choose **Optimization suggestions for all Regions**.

Suggestions to reduce latency for all Regions

To explore suggestions for reducing latency for all Regions where clients access your application, choose **Optimization suggestions for all Regions** to open a new dashboard

page. On this page, you can select different Regions to configure, with the option of including CloudFront as a configuration comparison, and then compare the times to first byte (TTFBs) for each selected configuration.

Then, for each comparison, you can also see a table with at a more granular level (by client location), with the average expected TTFB for each one.

Suggestions to reduce latency for top locations

Internet Monitor also provides suggestions for reducing application latency for your clients by specific location. When the table lists multiple suggestions for the same location, expand the city location for that row to see details.

Be aware that if you change a configuration to use a different Region or to use CloudFront, latency improvements can vary by client location. For example, latency might improve for some locations, but stay the same or worsen for others.

Suggestions to reduce latency by updating routing configurations

Note: These suggestions are only relevant for application traffic to Regional load balancers. The table is not shown for monitors that you create for CloudFront distributions or WorkSpaces resources.

With Internet Monitor, you can view information about latency toward AWS locations for IPv4 IP prefixes that access your application using different DNS resolvers (typically ISPs). Using this information, you can take steps to reduce latency for specific groups of users by routing a set of IP address prefixes, specified by a CIDR collection, to your endpoints in a Region that results in lower latency for your users. If you don't already have a CIDR collection for the prefixes, you can go to Amazon Route 53 to create one. Then, you can update your routing in Route 53 to route IP addresses in the collection to a specific Region.

If you want to create a CIDR collection for a set of IP address prefixes, you can easily do so by selecting a row or rows that includes the IP prefixes that you want, and then choosing **Add to CIDR collection**. Then, in the Route 53 console, you can configure a routing policy that routes IP addresses in the collection to the Region with lower latency for your application.

To learn more about IP-based routing in Route 53, see IP-based routing.

By viewing the suggestions on this page, you can start planning configurations and deployments that can improve performance for your clients. Note that you might see a dash (-) instead of a value in a column, when data is not available to display.

For more information about TTFB calculations, see AWS calculations for TTFB and latency. To review a specific example of how to improve performance, see Using Internet Monitor for a Better Gaming Experience.

Monitoring details in Internet Monitor (Configure page)

On the **Configure** page, you can see details about your monitor, including a list of resources that you monitor traffic for and the thresholds for when health events are triggered. You can also explore and compare values for the traffic percentage for your monitor, and its impact on how many city-networks are included for (monitored by) your monitor. Finally, you can view information about measurements that are published to an Amazon S3 bucket.

You can configure a monitor to change most options, such as the percentage of traffic to monitor. For more information, see Configure your monitor.

Monitor details

The **Monitor details** section includes basic information about your monitor, including the name, the percentage of traffic currently being monitored for your application, a city-networks maximum limit (if you've set one), and status information about the monitor.

The following explains the values that you might see for **Status** and **Status info** (data processing status).

Status	Description
Active	Monitor is created and active.
Pending	Monitor is currently being created and is not yet active.
Inactive	Monitor is created but has been set to inactive.
Error	Monitor is in an error state.

Status details (Data processing status)	Description

Status details (Data processing status)	Description
OK	Monitor is actively processing data.
Inactive	Monitor is inactive and is not processing data.
Collecting data	Monitor is actively collecting data.
Insufficient data	Monitor is actively processing data, but there aren't enough datapoints to produce insights.
Fault access CloudWatch	Monitor has encountered a problem delivering CloudWatch metrics data and log events.

Health event thresholds

In this section, you can see the current thresholds for health events that are configured for this monitor. If you haven't configured any custom thresholds, the values shown here are the default values.

By default, health events are not triggered based on local thresholds. If local health event thresholds would be useful for your Internet Monitor scenario, you can enable the option and specify the thresholds to use.

You can learn more about how health event thresholds work, and review the potential impact of adding local thresholds or changing existing thresholds. For more information, see Change health event thresholds.

Traffic coverage

In this section, you can explore options for the traffic coverage for your monitor. When you change the traffic percentage for a monitor, Internet Monitor monitors different amounts of application traffic. If you set a traffic percentage to less than 100% (100% is the default value), some portion of the city-networks that your clients use to access your application might not be monitored. By exploring the impact of different traffic percentage values, you can see how different values that you might set would impact your city-networks coverage.

The **Total monitored city-networks** graph shows you how many city-networks are currently monitored, and how many would be monitored if you set the traffic percentage to 100%. To view different traffic percentage values on the graph, select percentages in the drop-down menu.

After you explore the options, you can change the traffic percentage to monitor by choosing **Update monitoring coverage**.

If you want to set a maximum city-networks limit, at the top of the page, choose **Edit monitor**. Then, under **Advanced options**, set a maximum city-networks value.

Configure your monitor

As on every page in the Internet Monitor dashboard, you can choose **Edit monitor** to change options for your monitor, including adding or removing resources. For details about how to update the following configuration options, see the provided links.

View health event thresholds

In this section, you can see the current thresholds for health events that are configured for this monitor.

To update health thresholds, see Change health event thresholds.

View and evaluate traffic coverage

In this section, you can compare the effect of changing the percentage of traffic that you monitor for your application on the number of city-networks that are included (for monitoring) when you choose different percentage values.

You can also change the percentage of traffic that you monitor, or change the limit for the number of city-networks your monitor includes. To change the percentage of traffic, choose **Update monitoring coverage**.

For detailed steps and information, see Explore changing your application traffic percentage.

Configuration details for publishing internet measurements to Amazon S3

If you have configured Internet Monitor to publish internet measurements for your monitor to an Amazon S3 bucket, the information about your configuration is shown here.

To configure this option, see Publishing internet measurements to S3.

Exploring your data with CloudWatch tools and the Internet Monitor query interface

In addition to visualizing your performance and availability for your application with the Internet Monitor dashboard, there are several methods that you can use to dive deeper into the data that Internet Monitor generates for you. These methods include using CloudWatch tools with Internet Monitor data stored in CloudWatch Log files and using the Internet Monitor query interface. The tools that you can use include CloudWatch Logs Insights, CloudWatch Metrics, CloudWatch Contributor Insights, and Amazon Athena. You can use some or all these tools, as well as the dashboard, to explore Internet Monitor data, depending on your needs.

Internet Monitor aggregates CloudWatch metrics about traffic to your application and to each AWS Region, and includes data such as total traffic impact, availability, and round-trip time. This data is published to CloudWatch Logs and is also available to use with the Internet Monitor query interface. Details about geo-granularity and other aspects of the information available to explore for each one varies.

Internet Monitor publishes data for your monitor at 5 minute intervals, and then makes the data available in several ways. The following table lists scenarios for accessing Internet Monitor data, and describes features of the data that is collected for each one.

Feature	CloudWatch Logs	Export to S3	Query interface	CloudWatch dashboard
Enabled by default	Yes	No	Yes	Yes
Number of city-networks that data is collected for	Top 500 (see note below)	All	All	All
Data retention	User controlled	User controlled	30 days	30 days
Geo-granularities that data is collected for	All (city-network, metro+network, subdivision+networ	City-network	All (city-network, metro+network, subdivision+networ	All (city-network, metro+network, subdivision+networ

Feature	CloudWatch Logs	Export to S3	Query interface	CloudWatch dashboard
	k, country+network)		k, country+network)	k, country+network)
How to query and filter data	Use CloudWatch Logs Insights to explore Internet Monitor measurements	Use Amazon Athena to query internet measurements in Amazon S3 log files	Use the Internet Monitor query interface	Monitor and optimize with the Internet Monitor dashboard

Note: Top 500 measurements are captured for city-networks; top 250 for metro+networks, top 100 for subdivision+networks, top 50 for country+networks.

This chapter describes how to query and explore your data by using CloudWatch tools or the Internet Monitor query interface, together with examples for each method.

Contents

- Use CloudWatch Logs Insights to explore Internet Monitor measurements
- Use Contributor Insights to identify top locations and ISPs
- View Internet Monitor metrics or set alarms in CloudWatch Metrics
- Use Amazon Athena to query internet measurements in Amazon S3 log files
- Use the Internet Monitor query interface

Use CloudWatch Logs Insights to explore Internet Monitor measurements

You can use CloudWatch Logs Insights queries to filter a subset of logs for a specific city or geography (client location), client ASN (ISP), and AWS source location. Internet Monitor publishes granular measurements of availability and round-trip time to CloudWatch Logs that you can explore using CloudWatch Logs Insights.

To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.

The examples in this section can help you create CloudWatch Logs Insights queries to learn more about your own application traffic measurements and metrics. If you use these examples in CloudWatch Logs Insights, replace *monitorName* with your own monitor name.

View traffic optimization suggestions

On the **Traffic insights** tab in Internet Monitor, you can view traffic optimization suggestions, filtered by a location. To see the same information that is displayed in the **Traffic optimization suggestions** section on that tab, but without the location granularity filter, you can use the following CloudWatch Logs Insights query.

1. In the AWS Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/`*monitorName*`/byCity` and `/aws/internet-monitor/`*monitorName*`/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

```
fields @timestamp,
clientLocation.city as @city, clientLocation.subdivision as @subdivision,
 clientLocation.country as @country,
`trafficInsights.timeToFirstByte.currentExperience.serviceName` as @serviceNameField,
concat(@serviceNameField, ` (`, `serviceLocation`, `)`) as @currentExperienceField,
concat(`trafficInsights.timeToFirstByte.ec2.serviceName`, ` (`,
 `trafficInsights.timeToFirstByte.ec2.serviceLocation`, `)`) as @ec2Field,
`trafficInsights.timeToFirstByte.cloudfront.serviceName` as @cloudfrontField,
concat(`clientLocation.networkName`, ` (AS`, `clientLocation.asn`, `)`) as @networkName
| filter ispresent(`trafficInsights.timeToFirstByte.currentExperience.value`)
| stats avg(`trafficInsights.timeToFirstByte.currentExperience.value`) as @averageTTFB,
avg(`trafficInsights.timeToFirstByte.ec2.value`) as @ec2TTFB,
avg(`trafficInsights.timeToFirstByte.cloudfront.value`) as @cloudfrontTTFB,
sum(`bytesIn` + `bytesOut`) as @totalBytes,
latest(@ec2Field) as @ec2,
latest(@currentExperienceField) as @currentExperience,
latest(@cloudfrontField) as @cloudfront,
count(*) by @networkName, @city, @subdivision, @country
| display @city, @subdivision, @country, @networkName, @totalBytes, @currentExperience,
 @averageTTFB, @ec2, @ec2TTFB, @cloudfront, @cloudfrontTTFB
| sort @totalBytes desc
```

View internet availability and RTT (p50, p90, and p95)

To view the internet availability and round-trip time (p50, p90, and p95) for traffic, you can use the following CloudWatch Logs Insights query.

End user geography: Chicago, IL, United States

End user network (ASN): AS7018

AWS service location: US East (N. Virginia) Region

To view the logs, do the following:

1. In the AWS Management Console, navigate to CloudWatch Logs Insights.
2. For **Log Group**, select `/aws/internet-monitor/`*`monitorName`*`/byCity` and `/aws/internet-monitor/`*`monitorName`*`/byCountry`, and then specify a time range.
3. Add the following query, and then run the query.

The query returns all the performance data for users connecting from AS7018 in Chicago, IL towards US East (N. Virginia) Region over the selected time period.

```
fields @timestamp,
internetHealth.availability.experienceScore as availabilityExperienceScore,
internetHealth.availability.percentageOfTotalTrafficImpacted as
 percentageOfTotalTrafficImpacted,
internetHealth.performance.experienceScore as performanceExperienceScore,
internetHealth.performance.roundTripTime.p50 as roundTripTimep50,
internetHealth.performance.roundTripTime.p90 as roundTripTimep90,
internetHealth.performance.roundTripTime.p95 as roundTripTimep95
 | filter clientLocation.country == `United States`
 and clientLocation.city == `Chicago`
 and serviceLocation == `us-east-1`
 and clientLocation.asn == 7018
```

For more information, see Analyzing log data with CloudWatch Logs Insights.

Use Contributor Insights to identify top locations and ISPs

CloudWatch Contributor Insights can help you identify top client locations and ASNs (typically, internet service providers or ISPs) for your AWS application. Use the following sample Contributor Insights rules to get started with rules that are useful with Internet Monitor. For more information, see Create a Contributor Insights rule in CloudWatch.

To learn more about client location accuracy in Internet Monitor, see Geolocation information and accuracy in Internet Monitor.

> **Note**
>
> Internet Monitor stores internet measurements data every five minutes, so after you set up a Contributor Insights rule, you must adjust the period to five minutes to see a graph.

View top locations and ASNs impacted by an availability impact

To view top client locations and ASNs impacted by a drop in availability, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.availability.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View top client locations and ASNs impacted by a latency impact

To view top client locations and ASNs impacted by an increase in round-trip time (latency), you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.awsInternetHealth.performance.percentageOfTotalTrafficImpacted"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View top client locations and ASNs impacted by total percentage of traffic

To view top client locations and ASNs impacted by total percentage of traffic, you can use the following Contributor Insights rule in the Syntax editor. Replace *monitor-name* with your own monitor name.

```
{
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    },
    "AggregateOn": "Sum",
    "Contribution": {
        "Filters": [
```

```
            {
                "Match": "$.clientLocation.city",
                "IsPresent": true
            }
        ],
        "Keys": [
            "$.clientLocation.city",
            "$.clientLocation.networkName"
        ],
        "ValueOf": "$.percentageOfTotalTraffic"
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/aws/internet-monitor/monitor-name/byCity"
    ]
}
```

View Internet Monitor metrics or set alarms in CloudWatch Metrics

You can view or set alarms on Internet Monitor metrics by using CloudWatch alarms and CloudWatch Metrics in the CloudWatch console. Internet Monitor publishes metrics to your account, including metrics for performance, availability, round-trip time, and throughput (bytes per second). To find all metrics for your monitor, in the CloudWatch Metrics dashboard, see the custom namespace `AWS/InternetMonitor`.

To see examples for using several of these metrics to help determine values to choose for a city-networks maximum limit for your monitor, see Choosing a city-network maximum value. To learn more about setting alarms for Internet Monitor, see Create alarms with Internet Monitor.

Metrics are aggregated across all internet traffic to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories in the monitor, and to all traffic to each AWS Region and internet edge location that is monitored. Regions are defined by the service location, which can either be all locations or a specific Region, such as `us-east-1`.

Note: *city-networks* are pairs of client locations and the ASNs the clients use (typically internet service providers or ISPs).

Internet Monitor provides the following metrics.

Metric	Description
PerformanceScore	A performance score represents the estimated percentage of traffic that is not seeing a performance drop.
AvailabilityScore	An availability score represents the estimated percentage of traffic that is not seeing an availability drop.
BytesIn	Bytes transferred in for your application internet traffic at all application city-networks.
BytesOut	Bytes transferred out for your application internet traffic at all application city-networks.
BytesInMonitored	Bytes transferred in for your application internet traffic at monitored city-networks.
BytesOutMonitored	Bytes transferred out for your application internet traffic at monitored city-networks.
Round-trip time (RTT)	Round-trip time between the AWS Regions, ASNs (typically internet service providers or ISPs), and locations (such as cities) specific to your VPCs, Network Load Balancers, CloudFront distributions, or WorkSpaces directories.
CityNetworksMonitored	The number of city-networks Internet Monitor monitored for your application internet traffic. This is never more than the upper limit that you set as the maximum city-networks for the monitor.
TrafficMonitoredPercent	The percentage of total application internet traffic for this monitor that is represented (included) by the city-networks that Internet Monitor is monitoring. This is less than 100

Metric	Description
	(that is, less than 100%) if clients access your application in more city-networks than the maximum city-networks limit that you have set for the monitor.
CityNetworksFor100PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 100% of your application internet traffic in Internet Monitor.
CityNetworksFor99PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 99% of your application internet traffic in Internet Monitor.
CityNetworksFor95PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 95% of your application internet traffic in Internet Monitor.
CityNetworksFor90PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 90% of your application internet traffic in Internet Monitor.
CityNetworksFor75PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 75% of your application internet traffic in Internet Monitor.
CityNetworksFor50PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 50% of your application internet traffic in Internet Monitor.

Metric	Description
CityNetworksFor25PercentTraffic	The number that you should set your city-networks maximum limit to if you want to monitor 25% of your application internet traffic in Internet Monitor.

For more information, see Metrics in Amazon CloudWatch.

Use Amazon Athena to query internet measurements in Amazon S3 log files

You can use Amazon Athena to query and view the internet measurements that Internet Monitor publishes to an Amazon S3 bucket. There's an option in Internet Monitor to publish internet measurements for your application to an S3 bucket for internet-facing traffic for your monitored city-networks (client locations and ASNs, typically internet service providers or ISPs). Regardless of whether you choose to publish measurements to S3, Internet Monitor automatically publishes internet measurements to CloudWatch Logs every five minutes for the top 500 (by traffic volume) city-networks for each monitor.

This chapter includes steps for how to create a table in Athena for internet measurements located in an S3 log file, and then provides example queries to see different views of the measurements. For example, you can query for your top 10 impacted city-networks by latency impact.

Using Amazon Athena to create a table for internet measurements in Internet Monitor

To start using Athena with your Internet Monitor S3 log files, you first create a table for the internet measurements.

Follow the steps in this procedure to create a table in Athena based on the S3 log files. Then, you can run Athena queries on the table, such as these example internet measurements queries, to get information about your measurements.

To create an Athena table

1. Open the Athena console at https://console.aws.amazon.com/athena/.
2. In the Athena query editor, enter a query statement to generate a table with Internet Monitor internet measurements. Replace the value for the LOCATION parameter with the location of S3 bucket where your Internet Monitor internet measurements are stored.

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION
's3://amzn-s3-demo-bucket/bucket_prefix/AWSLogs/account_id/
internetmonitor/AWS_Region/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

3. Enter a statement to create a partition to read the data. For example, the following query creates a single partition for a specified date and location:

```
ALTER TABLE internet_measurements
ADD PARTITION (year = 'YYYY', month = 'MM', day = 'dd')
LOCATION
's3://amzn-s3-demo-bucket/bucket_prefix/AWSLogs/account_id/
internetmonitor/AWS_Region/YYYY/MM/DD';
```

4. Choose **Run**.

Example Athena statements for internet measurements

The following is an example of a statement to generate a table:

```
CREATE EXTERNAL TABLE internet_measurements (
    version INT,
    timestamp INT,
    clientlocation STRING,
    servicelocation STRING,
    percentageoftotaltraffic DOUBLE,
    bytesin INT,
    bytesout INT,
```

```
    clientconnectioncount INT,
    internethealth STRING,
    trafficinsights STRING
)
PARTITIONED BY (year STRING, month STRING, day STRING)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/'
TBLPROPERTIES ('skip.header.line.count' = '1');
```

The following is an example of a statement to create a partition to read the data:

```
ALTER TABLE internet_measurements
ADD PARTITION (year = '2023', month = '04', day = '07')
LOCATION 's3://internet-measurements/TestMonitor/AWSLogs/1111222233332/internetmonitor/
us-east-2/2023/04/07/'
```

Sample Amazon Athena queries to use with internet measurements in Internet Monitor

This section includes example queries that you can use with Amazon Athena to get information about your application's internet measurements published to Amazon S3.

Query your top 10 impacted (by total percentage of traffic) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(percentageoftotaltraffic) as percentageoftotaltraffic
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageoftotaltraffic desc
limit 10
```

Query your top 10 impacted (by availability) client locations and ASNs

Run this Athena query to return your top 10 impacted (by total percentage of traffic) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
```

```
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.availability.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

Query your top 10 impacted (by latency) client locations and ASNs

Run this Athena query to return your top 10 impacted (by latency impact) city-networks—that is, client locations and ASNs, typically internet service providers.

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.networkname') as networkName,
    sum(
        cast(
            json_extract_scalar(
                internetHealth,
                '$.performance.percentageoftotaltrafficimpacted'
            )
        as double )
    ) as percentageOfTotalTrafficImpacted
FROM internet_measurements
GROUP BY json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.networkname')
ORDER BY percentageOfTotalTrafficImpacted desc
limit 10
```

Query traffic highlights for your client locations and ASNs

Run this Athena query to return traffic highlights, including availability score, performance score, and time to first byte for your city-networks—that is, client locations and ASNs, typically internet service providers. .

```
SELECT json_extract_scalar(clientLocation, '$.city') as city,
    json_extract_scalar(clientLocation, '$.subdivision') as subdivision,
    json_extract_scalar(clientLocation, '$.country') as country,
    avg(cast(json_extract_scalar(internetHealth, '$.availability.experiencescore') as
 double)) as availabilityScore,
    avg(cast(json_extract_scalar(internetHealth, '$.performance.experiencescore') as
 double)) performanceScore,
    avg(cast(json_extract_scalar(trafficinsights,
 '$.timetofirstbyte.currentexperience.value') as double)) as averageTTFB,
    sum(bytesIn) as bytesIn,
    sum(bytesOut) as bytesOut,
    sum(bytesIn + bytesOut) as totalBytes
FROM internet_measurements
where json_extract_scalar(clientLocation, '$.city') != 'N/A'
GROUP BY
json_extract_scalar(clientLocation, '$.city'),
    json_extract_scalar(clientLocation, '$.subdivision'),
    json_extract_scalar(clientLocation, '$.country')
ORDER BY totalBytes desc
limit 100
```

For more information about using Athena, see the Amazon Athena User Guide.

Use the Internet Monitor query interface

An option for understanding more about internet traffic for your AWS application is to use the Internet Monitor *query interface*. To use the query interface, you create a query with data filters that you choose, and then run the query to return a subset of your Internet Monitor data. Exploring the data that the query returns can give you insights into how your application is performing on the internet.

You can query and explore all the metrics that Internet Monitor captures with your monitor, including availability and performance scores, bytes transferred, round-trip times, and time to first byte (TTFB).

Internet Monitor uses the query interface to provide the data that you can explore in the Internet Monitor console dashboard. By using search options in the dashboard—on the **Analyze** page or the **Optimize** page—you can query and filter internet data for your application.

If you'd like more flexibility to explore and filter your data than the dashboard provides, you can use the query interface yourself, by using Internet Monitor API operations with the AWS Command Line Interface or with an AWS SDK. This section introduces the types of queries that you can use

with the query interface. It also describes the filters that you can specify to create a subset of data, to get insights about internet traffic for your application.

Topics

- How to use the query interface
- Query examples
- Get query results
- Troubleshooting

How to use the query interface

You create a query with the query interface by choosing a *query type*, and then specifying filter values, to return a specific desired subset of your log file data. Then, you can work with the data subset, to further filter and sort, create reports, and so on.

The query process works like this:

1. When you run a query, Internet Monitor returns a `query ID` that is unique to the query. This section describes the query types that are available, and options for filtering data in queries. To understand how this works, you can also review the section on query examples.
2. You specify the query ID with your monitor name with the GetQueryResults API operation to return data results for the query. Each query type returns a different set of data fields. To learn more, see Get query results.

The query interface provides the following query types. Each query type returns a different set of information about your traffic from the log files, as shown.

- **Measurements:** Provides availability score, performance score, total traffic, and round-trip times, at 5 minute intervals.
- **Top locations:** Provides availability score, performance score, total traffic, and time to first byte (TTFB) information, for the top location and ASN combinations that you're monitoring, by traffic volume.
- **Top locations details:** Provides TTFB for Amazon CloudFront, your current configuration, and the best performing Amazon EC2 configuration, at 1 hour intervals.
- **Overall traffic suggestions:** Provides TTFB, using a 30-day weighted average, for all traffic in each AWS location that is monitored.

- **Overall traffic suggestions details:** Provides TTFB, using a 30-day weighted average, for each top location, for a proposed AWS location.
- **Routing suggestions:** Provides the predicted average round-trip time (RTT) from an IP prefix toward an AWS location for a DNS resolver. The RTT is calculated at one hour intervals, over a one hour period.

You can filter the data more by using specific criteria. With most query types, except routing suggestions, you can filter by specifying one or more of the following criteria:

- **AWS location:** For AWS location, you can specify CloudFront or an AWS Region, such as `us-east-2`.
- **ASN:** Specify the autonomous system number (ASN) of a DNS resolver (typically, an ISP), for example, 4225.
- **Client location:** For location, specify a city, metro, subdivision, or country.
- **Proposed AWS location:** Specify an AWS Region, such as `us-east-2`, or an AWS Local Zone. You can use this filter with the overall traffic suggestions details query type.
- **Geo:** Specify `geo` for some queries. This is required for queries that use the `Top locations` query type, but not allowed for other query types. To understand when to specify `geo` for filter parameters, see the query examples section.

For the routing suggestions query type, you can filter the data more by specifying one or more of the following criteria:

- **Current AWS location:** Specify an AWS Region, such as `us-east-2`.
- **Proposed AWS location:** Specify an AWS Region, such as `us-east-2`, or an AWS Local Zone.
- **IPv4 prefix:** Specify an IPv4 prefix in the standard format, similar to `192.0.2.0/24`.
- **Monitor ARN:** Specify the ARN for a specific monitor.
- **DNS resolver IP:** Specify the IP address of a DNS resolver.
- **DNS resolver ISP:** Specify the name of a DNS resolver (typically an ISP), for example, `Cloudflare`.
- **DNS resolver ASN:** Specify the autonomous system number (ASN) of a DNS resolver, for example, 4225.

The operators that you can use for filtering your data are EQUALS and NOT_EQUALS. For details about filtering parameters, see the FilterParameter API operation.

To see details about the query interface operations, see the following API operations in the Internet Monitor API Reference Guide:

- To create and run a query, see the StartQuery API operation.
- To stop a query, see the StopQuery API operation.
- To return data for a query that you've created, see the GetQueryResults API operation.
- To retrieve the status of a query, see the GetQueryStatus API operation.

Query examples

To create a query that you can use to retrieve a filtered set of data from your monitor's log file, you use the StartQuery API operation. You specify a query type and filter parameters for the query. Then, when you use the Internet Monitor query interface API operation to get query results using the query, it will retrieve the subset of your data that you want to work with.

To illustrate how query types and filter parameters work, let's look at some examples.

Example 1

Let's say that you want to retrieve all of your monitor's log file data for a specific country, except for one city. The following example shows filter parameters for a query that you could create with the StartQuery operation for this scenario.

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "MEASUREMENTS"
   FilterParameters: [
      {
       Field: "country",
       Operator: "EQUALS",
       Values: ["Germany"]
      },
      {
       Field: "city",
       Operator: "NOT_EQUALS",
       Values: ["Berlin"]
```

```
      },
    ]
}
```

Example 2

As another example, let's say that you want to see your top locations by metropolitan area. You could use the following example query for this scenario.

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["metro"]
      },
    ]
}
```

Example 3

Now, let's say that you want to see the top city-network combinations in the Los Angeles metro area. To do this, specify `geo=city`, and then set `metro` to Los Angeles. Now, the query returns the top city-networks in the Los Angeles metro area instead of the top metro+networks overall.

Here's the example query that you could use:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATIONS"
   FilterParameters: [
      {
       Field: "geo",
       Operator: "EQUALS",
       Values: ["city"]
      },
      {
```

```
        Field: "metro",
        Operator: "EQUALS",
        Values: ["Los Angeles"]
       }
    ]
}
```

Example 4

Next, let's say that you want to retrieve TTFB data for a specific subdivision (for example, a U.S. state).

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "TOP_LOCATION_DETAILS"
   FilterParameters: [
      {
       Field: "subdivision",
       Operator: "EQUALS",
       Values: ["California"]
      },
    ]
}
```

Example 5

Now, let's say that you want to retrieve TTFB data for every location where your application has client traffic.

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS"
   FilterParameters: []
}
```

```
Results:
[us-east-1, 40, us-west-2, 30],
[us-east-1, 40, us-west-1, 35],
[us-east-1, 40, us-east-1, 44],
[us-east-1, 40, CloudFront, 22],
...
[us-east-2, 44, us-west-2, 30],
[us-east-2, 44, us-west-1, 35],
...
```

Example 6

Let's say that you want to retrieve TTFB data for a specific new AWS Region.

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "OVERALL_TRAFFIC_SUGGESTIONS_DETAILS"
   FilterParameters: [
      {
       Field: "proposed_aws_location",
       Operator: "EQUALS",
       Values: ["us-west-2"]
      },
   ]
}

Results:
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-east-1, 40, 350,
 350, us-west-2, 45]
[San Jose, San Jose-Santa Clara, California, United States, 7922, us-west-1, 35, 450,
 450, us-west-2, 45]
```

Example 7

A final example is to retrieve data for specific DNS resolvers.

The following is an example query for this scenario:

```
{
   MonitorName: "TestMonitor"
```

```
   StartTime: "2023-07-12T20:00:00Z"
   EndTime: "2023-07-12T21:00:00Z"
   QueryType: "ROUTING_SUGGESTIONS"
   FilterParameters: [
      {
       Field: "proposed_aws_location",
       Operator: "EQUALS",
       Values: ["us-east-1"]
      },
   ]
}

Results:
[162.158.180.245, 13335, Cloudflare, [5.4.0.0/14], us-east-2, 200.0, us-east-1, 160.0]
[162.158.180.243, 13313, Cloudflare, [5.4.0.0/10], us-east-2, 150.0, us-east-1, 125.0]
```

Get query results

After you define a query, you can return a set of results with the query by running another Internet Monitor API operation, GetQueryResults. When you run `GetQueryResults`, you specify the query ID for the query that you've defined, along with the name of your monitor. `GetQueryResults` retrieves data for the specified query into a result set.

When you run a query, make sure that the query has finished running before you use `GetQueryResults` to look at the results. You can determine if the query has completed by using the GetQueryStatus API operation. When the `Status` for the query is SUCCEEDED, you can go ahead with reviewing the results.

When your query completes, you can use the following information to help you review the results. Each query type that you use to create a query includes a unique set of data fields from the log files, as described in the following list:

Measurements

The `measurements` query type returns the following data:

`timestamp, availability, performance, bytes_in, bytes_out, rtt_p50, rtt_p90, rtt_p95`

Top locations

The `top locations` query type groups data by location, and provides the data averaged over the time period. The data that it returns includes the following:

`aws_location, city, metro, subdivision, country, asn, availability, performance, bytes_in, bytes_out, current_fbl, best_ec2, best_ec2_region, best_cf_fbl`

Note that `city`, `metro`, and `subdivision` are only returned if you choose that location type for the `geo` field. The following location fields are returned, depending on the location type that you specify for `geo`:

```
city = city, metro, subdivision, country
metro = metro, subdivision, country
subdivision = subdivision, country
country = country
```

Top locations details

The `top locations details` query type returns data grouped hour by hour. The query returns the following data:

`timestamp, current_service, current_fbl, best_ec2_fbl, best_ec2_region, best_cf_fbl`

Overall traffic suggestions

The `overall traffic suggestions` query type returns data grouped hour by hour. The query returns the following data:

`current_aws_location, proposed_aws_location, average_fbl, traffic, optimized_traffic_excluding_cf, optimized_traffic_including_cf`

Overall traffic suggestions details

The `overall traffic suggestions details` query type returns data grouped hour by hour. The query returns the following data:

`aws_location, city, metro, subdivision, country, asn, traffic, current_aws_location, fbl_data`

Routing suggestions

The `routing suggestions` query type returns data grouped hour by hour. The query returns the following data:

```
dns_resolver_ip, dns_resolver_asn, dns_resolver_isp, ipv4_prefixes,
current_aws_location, current_latency, proposed_aws_location,
proposed_latency
```

When you run the `GetQueryResults` API operation, Internet Monitor returns the following in the response:

- A *data string array* that contains the results that the query returns. The information is returned in arrays that are aligned with the `Fields` field, also returned by the API call. Using the `Fields` field, you can parse the information from the `Data` repository and then further filter or sort it for your purposes.
- An *array of fields* that lists the fields that the query returned data for (in the `Data` field response). Each item in the array is a name-datatype pair, such as `availability_score-float`.

Troubleshooting

If errors are returned when you use query interface API operations, verify that you have the required permissions to use Internet Monitor. Specifically, make sure that you have the following permissions:

```
internetmonitor:StartQuery
internetmonitor:GetQueryStatus
internetmonitor:GetQueryResults
internetmonitor:StopQuery
```

These permissions are included in the recommended AWS Identity and Access Management policy to use the Internet Monitor dashboard in the console. For more information, see AWS managed policies for Internet Monitor.

Add a monitor using other AWS services

A simple way to add monitoring with Internet Monitor is to choose to create a monitor when you add a supported resource when you create a resource—or use monitoring for the resource—in console.

Resources with an integrated option for adding Internet Monitor include the following:

- VPCs
- Network Load Balancers
- Amazon CloudFront distributions

The following sections provide more information about Internet Monitor integrations in the service consoles for supported resources.

Contents

- Add a monitor with a Network Load Balancer
- Add an Internet Monitor monitor with Amazon VPC
- Add an Internet Monitor monitor with CloudFront

Add a monitor with a Network Load Balancer

When you create a Network Load Balancer in the AWS Management Console, you can optionally choose to also set up monitoring for traffic to and from the Network Load Balancer using a monitor in Internet Monitor. You can add the Network Load Balancer to an existing monitor, or you can opt to create a new monitor for your Network Load Balancer traffic.

By using Internet Monitor with your Network Load Balancer, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically, internet service providers). Internet Monitor also determines when there are anomalies in performance and availability, and then creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

Important

To create a monitor, or add a Network Load Balancer to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a Network Load Balancer to an existing monitor

When you create the Network Load Balancer in the AWS Management Console, you can choose to have Internet Monitor add the new Network Load Balancer to an existing monitor. Under **Integrations**, choose Internet Monitor, and then choose **Add monitor**. Choose **Select an existing monitor**, and then enter a monitor name. Or choose **View monitors** to go to the Internet Monitor console, and then scroll down to see a list of available monitors.

After you add the Network Load Balancer to a monitor, wait a few minutes, and then metrics for traffic to and from the load balancer will start being shown on the Internet Monitor console. To learn more about the **Status** and **Data processing status** values, see Monitoring details in Internet Monitor (Configure page).

You can edit the monitor at any time, to remove the load balancer or add another Network Load Balancer, or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the Network Load Balancer from the monitor, traffic from clients to that load balancer is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a Network Load Balancer

Under **Integrations**, choose Internet Monitor, and then choose **Monitor resource traffic**. Choose **Create a new monitor**, and then enter a monitor name. Leave the default traffic percentage to monitor, 100%, or specify a custom percentage, and then choose **Create monitor**.

After you create the monitor, wait a few minutes, and then metrics for traffic to and from the Network Load Balancer will start being shown on the Internet Monitor console. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor.

Stop monitoring a Network Load Balancer

If you'd like to stop monitoring your Network Load Balancer resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the Network Load Balancer to remove, and then choose **Remove**.
7. Choose **Update**.

Add an Internet Monitor monitor with Amazon VPC

When you create a Amazon Virtual Private Cloud VPC in the AWS Management Console, you can optionally choose to also set up monitoring for it in Internet Monitor. You can add the VPC to an existing monitor, or you can opt to create a new monitor for the VPC in the Amazon VPC console.

By using Internet Monitor with your VPC, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

> **Important**
>
> To create a monitor, or add a VPC to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a VPC to an existing monitor

You can choose to have Internet Monitor add a new VPC to an existing monitor for you when you create the VPC in the AWS Management Console. After you add the VPC, wait a few minutes, and then metrics for the VPC will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the VPC or add another VPC or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the VPC from the monitor, traffic from clients to that VPC is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a VPC

If you opt to create a monitor for a VPC, the **Create monitor** wizard walks you through the steps. You add the VPC as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor

Stop monitoring a VPC

If you'd like to stop monitoring your VPC resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.

4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the VPC to remove, and then choose **Remove**.
7. Choose **Update**.

Add an Internet Monitor monitor with CloudFront

On the metrics dashboard for a distribution in Amazon CloudFront console, you can set up additional monitoring for a distribution in Internet Monitor. You can add the distribution to an existing monitor, or you can create a new monitor for the distribution.

By using Internet Monitor with your CloudFront distribution, you can view and evaluate measurements and metrics about availability, performance, monitored bytes transferred, and round-trip times that are specific to your application's client locations and ASNs (typically internet service providers). Internet Monitor also determines when there are anomalies in performance and availability and creates health events in your monitor, which you can choose to be notified about. To learn more about how you can use a monitor to manage and improve your clients' experience with your application, see Use a monitor in Internet Monitor.

> **Important**
>
> To create a monitor, or add a distribution to an existing monitor, you must have the correct permissions in place. For more information, see Identity and Access Management for Internet Monitor.

Add a distribution to an existing monitor

You can choose to have Internet Monitor add a distribution to an existing monitor directly from the CloudFront metrics dashboard in the AWS Management Console. After you add the distribution, wait a few minutes, and then metrics for the distribution will start being shown on the Internet Monitor console.

You can edit the monitor at any time, to remove the distribution or add another distribution or other resources. You can also change the percentage of traffic that you're monitoring, or make other changes. If you choose to remove the distribution from the monitor, traffic from clients to that distribution is no longer monitored by Internet Monitor.

To learn more about updating a monitor, see Edit a monitor in Internet Monitor.

Create a monitor for a distribution

If you opt to create a monitor for a distribution, the **Create monitor** wizard walks you through the steps. You add the distribution as a monitored resource when you create the monitor. If you like, you can also choose a percentage of client traffic that you want to monitor for your application (the default is 100%).

You can learn more by reviewing the information in Step 1: Create a monitor.

Pricing

With Internet Monitor, you pay only for what you use. Pricing for Internet Monitor has two components: a per monitored resource fee and a per city-network fee. A city-network is the location that clients access your application resources from and the network (an ASN, such as an internet service provider or ISP) that clients access the resources through.

For more information, including pricing examples, see Pricing for Internet Monitor.

Stop monitoring a distribution

If you'd like to stop monitoring your distribution resource with Internet Monitor, do the following in the Internet Monitor console:

To remove a resource from a monitor

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation pane, under **Network Monitoring**, choose **Internet monitors**.
3. Choose your monitor, and then choose the **Action** menu.
4. Choose **Update monitor**.
5. Under **Added resources**, choose **Remove resources**.
6. Choose the distribution to remove, and then choose **Remove**.
7. Choose **Update**.

Create alarms with Internet Monitor

You can create Amazon CloudWatch alarms based on Internet Monitor metrics, just as you can for other Amazon CloudWatch metrics.

For example, you can create an alarm based on the Internet Monitor metric `PerformanceScore`, and configure it to send a notification when the metric is lower than a value that you choose. You configure alarms for Internet Monitor metrics following the same guidelines as for other CloudWatch metrics.

Following are the example Internet Monitor metrics that you might choose to create an alarm for:

- **PerformanceScore**
- **AvailabilityScore**
- **RoundtripTime**

To see all the metrics available for Internet Monitor, see View Internet Monitor metrics or set alarms in CloudWatch Metrics.

The following procedure provides an example of setting an alarm on **PerformanceScore** by navigating to the metric in the CloudWatch dashboard. Then, you follow the standard CloudWatch steps to create an alarm based on a threshold that you choose, and set up a notification or choose other options.

To create an alarm for PerformanceScore in CloudWatch Metrics

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, and then choose **Classic metrics**.
3. Filter for Internet Monitor by choosing `AWS/InternetMonitor`.
4. Choose **MeasurementSource, MonitorName**.
5. In the list, select **PerformanceScore**.
6. On the **GraphedMetrics** tab, under **Actions**, choose the bell icon to create an alarm based on a static threshold.

Now, follow the standard CloudWatch steps to choose options for the alarm. For example, you can choose to be notified by an Amazon SNS message if **PerformanceScore** is below a specific threshold number. Alternatively, or in addition, you can add the alarm to a dashboard.

Keep in mind the following:

- Internet Monitor metrics are typically calculated and published within 20 minutes.

- When you create an alarm based on Internet Monitor metrics, make sure that you take into account the short delay before publication when you set an alarm's lookback period. We recommend that you configure **Evaluation Periods** with lookback period that is a minimum of 25 minutes.

To learn more about using CloudWatch alarms with Internet Monitor, see the following blog post: Using Internet Monitor for enhanced internet observability.

For more information about options when you create a CloudWatch alarm, see Create a CloudWatch alarm based on a static threshold.

Using Internet Monitor with Amazon EventBridge

Health events that Internet Monitor creates for networking issues are published with Amazon EventBridge, so that you can send notifications about a degradation in end users' experience for your application. Both overall (global) health events and local health events are published to EventBridge. Global health events have an impact type of `AVAILABILITY` or `PERFORMANCE`, and local health events have an impact type of `LOCAL_AVAILABILITY` or `LOCAL_PERFORMANCE`.

To use EventBridge to work with Internet Monitor health events, follow the guidance here.

To set up a rule for Internet Monitor in EventBridge

1. In the AWS Management Console, in EventBridge, choose **Rules**, then enter a name and a description. Create the rule on the **Default** event bus.
2. In Step 2, select **Other** for the event source, and then, under **Event pattern**, match the following source.

```
{
  "source": ["aws.internetmonitor"]
}
```

3. In Step 3, for the target, select **AWS Service** and **CloudWatch Logs Group**, then select an existing log group or create a new one.
4. Add any desired tags, and then create the rule. This should populate your selected CloudWatch Logs Group with events from EventBridge.

Filtering health events by impact type

You can use event patterns to filter health events by impact type. This is useful if you want to receive only global health events, only local health events, or both.

Receive only global health events:

```
{
  "source": ["aws.internetmonitor"],
  "detail": {
    "impactType": ["AVAILABILITY", "PERFORMANCE"]
  }
}
```

Receive only local health events:

```
{
  "source": ["aws.internetmonitor"],
  "detail": {
    "impactType": ["LOCAL_AVAILABILITY", "LOCAL_PERFORMANCE"]
  }
}
```

Receive all health events (default):

```
{
  "source": ["aws.internetmonitor"]
}
```

For more information about how EventBridge rules work with event patterns, see Amazon EventBridge event patterns in the Amazon EventBridge User Guide.

Troubleshoot CloudWatch logs and metrics access errors

To support some features, Internet Monitor must interact with certain Amazon CloudWatch resources, including logs and metrics. If Internet Monitor can't access the CloudWatch resources that it requires access to, Internet Monitor sets a status code of `FAULT_ACCESS_CLOUDWATCH` for the monitor.

There are several reasons that your monitor might have the state `FAULT_ACCESS_CLOUDWATCH`. The following sections list possible causes for these errors, and suggested troubleshooting steps.

Internet Monitor couldn't access CloudWatch logs in your account

Internet Monitor publishes diagnostic logs about your monitored application traffic. It publishes these logs to log groups in CloudWatch Logs in the following location: `/aws/internet-monitor/monitor_name/[byCity|byMetro|bySubdivision|byCountry]`. Internet Monitor was unable to access these log groups.

Error states and potential solutions:

- **PutLogEvents throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's logs to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **Log group not found:** Disable, and then re-enable your monitor. Enabling a monitor restarts log group creation, which might correct the problem.
- **PutLogEvents access denied error:** Contact AWS support for assistance.
- **PutLogEvents unknown or general error:** Contact AWS support for assistance.

Internet Monitor couldn't access CloudWatch metrics in your account

Internet Monitor delivers specific CloudWatch metrics about the application traffic that is tracked by a monitor. An error occurred when Internet Monitor tried to deliver these metrics to CloudWatch.

Error states and potential solutions:

- **PutMetricData throttling error:** The Internet Monitor service might have been throttled when it tried to publish your monitor's metrics to CloudWatch. Review the throttling limits for your account, and, if necessary, request an increase in the limit.
- **PutMetricData access denied error:** Contact AWS support for assistance.
- **PutMetricData unknown or general error:** Contact AWS support for assistance.

Data protection and data privacy with Internet Monitor

The AWS shared responsibility model applies to data protection and data privacy in Internet Monitor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS cloud. You are responsible for maintaining control over your content that

is hosted on this infrastructure. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see The AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog. For more resources about complying with GDPR requirements, see the General Data Protection Regulation (GDPR) Center.

We strongly recommend that you never put sensitive identifying information, such as your end users' account numbers, email addresses, or other personal information, into free-form fields. Any data that you enter into Internet Monitor or other services might be included in diagnostic logs.

Identity and Access Management for Internet Monitor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Internet Monitor resources. IAM is an AWS service that you can use with no additional charge.

Important

Internet Monitor resource changes on July 8, 2024
If you created IAM policies that included Internet Monitor resources before July 8, 2024, be aware of the following change to Internet Monitor resources and resource types:

- Resource-level permissions for the **GetHealthEvent** action are now supported only on the **Monitor** resource type. The permissions are not supported on the **HealthEvent** resource.

To see more information about the actions, resources, and condition keys that you can specify in policies to manage access to AWS resources in Internet Monitor, see Actions, resources, and condition keys for Internet Monitor.

Contents

- Upgrade IAM policies to IPv6
- How Internet Monitor works with IAM
- Cross-service confused deputy prevention
- AWS managed policies for Internet Monitor
- Service-linked role for Internet Monitor

Upgrade IAM policies to IPv6

Internet Monitor customers use IAM policies to set an allowed range of IP addresses, to prevent any IP addresses outside the configured range from being able to access Internet Monitor APIs.

The *internetmonitor.region.api.aws* endpoint, where you access Internet Monitor APIs, is being upgraded to support dual-stack (IPv4 and IPv6).

IP address filtering policies that are not updated to handle IPv6 addresses might result in clients losing access to Internet Monitor APIs.

Customers impacted by the upgrade to include IPv6

Customers who are using dual-stack with policies that contain the *aws:sourceIp* filter are impacted by this upgrade. Dual-stack means that the network supports both IPv4 and IPv6.

If you use dual-stack, you must update your IAM policies that are currently configured with IPv4 format addresses to include IPv6 format addresses.

The following summarizes recommended actions, depending on your scenario. To confirm the endpoint that your SDK uses, see Identify the Internet Monitor endpoint used by your code.

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.amazonaws.com` (not dual-stack)	Yes	To restrict access to IPv4 only, take no further action. Or, if you anticipate that you will need IPv6 support in the future, you can take action to be compatible with both IPv4 and IPv6. To ensure future compatibility, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoi`

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
		`nt=true` . For more informati on, see Dual-stack and FIPS endpoints. If you choose to use both IPv4 and IPv6, you must also update the IP address filtering condition (`aws:sourceIp`) in your IAM policies to include IPv6 addresses.
`internetmonitor.region.amazonaws.com` (not dual-stack)	No	To restrict access to IPv4 only, take no further action. Or, if you anticipate that you will need IPv6 support in the future, you can take action to be compatible with both IPv4 and IPv6. To ensure future compatibility, on or after November 1, 2024, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true` . For more information, see Dual-stack and FIPS endpoints.

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.api.aws`	Yes	To ensure future compatibility with both IPv4 and IPv6, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true`. For more information, see Dual-stack and FIPS endpoints. When you make the change to use both IPv4 and IPv6, you must also update the IP address filtering condition (`aws:sourceIp`) in your IAM policies to include IPv6 addresses. If you instead want to restrict access to IPv4 only, set `useDualstackEndpoint=false`. For more information, see Dual-stack and FIPS endpoints.

Endpoint	Using IAM policy with `aws:sourceIp` condition?	Recommended action
`internetmonitor.region.api.aws`	No	To ensure future compatibility with both IPv4 and IPv6, update your SDK, and then update your application to use the dual-stack endpoint by setting `useDualstackEndpoint=true`. For more information, see Dual-stack and FIPS endpoints. If you instead want to restrict access to IPv4 only, set `useDualstackEndpoint=false`. For more information, see Dual-stack and FIPS endpoints.

For help with access issues, contact Support.

What is IPv6?

IPv6 is the next generation IP standard intended to eventually replace IPv4. IPv4 uses a 32-bit addressing scheme, to support 4.3 billion devices. IPv6 instead uses 128-bit addressing, to support approximately 340 trillion trillion trillion (or 2 to the 128th power) devices.

The following are examples of IPv6 addresses:

```
2001:cdba:0000:0000:0000:0000:3257:9652
2001:cdba:0:0:0:0:3257:9652
2001:cdba::3257:965
```

IPv6 offers a larger address space, improved routing efficiency, and better support for new internet services. By updating to dual-stack and supporting IPv6, Internet Monitor enables improved

performance and scalability. Follow the steps in this section to update your configurations and take advantage of dual-stack support.

Identify the Internet Monitor endpoint used by your code

If you use an Internet Monitor SDK, start by verifying which endpoint your code is using: the IPv4 endpoint or the dual-stack (IPv4 and IPv6) endpoint. If you don't use an SDK with Internet Monitor, you can skip this section.

You can run the following code example to determine the Internet Monitor endpoint that you're using. For this example, we're using the Internet Monitor SDK for Go in the US East (N. Virginia) Region.

```
package main

import (
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/internetmonitor"
)

func main() {
    // Create a new session with the default configuration
    sess := session.Must(session.NewSession(&aws.Config{
        Region: aws.String("us-east-1"),
    }))

    // Create a new Internet Monitor client
    internetMonitorClient := internetmonitor.New(sess)

    // Get the endpoint URL
    endpoint := internetMonitorClient.Endpoint

    fmt.Printf("Internet Monitor endpoint URL: %s\n", endpoint)
}
```

When you run this code, it returns the Internet Monitor endpoint. If you see the following response, you're using the Internet Monitor domain that supports only IPv4. You can tell because the format of the endpoint URL includes `amazonaws.com`.

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.amazonaws.com
```

If you see the following response instead, then you're using the domain which is being upgraded to support dual-stack (IPv4 and IPv6). Here, you can tell because the endpoint URL includes `api.aws`. However, note that until the upgrade is complete, this endpoint supports only IPv4.

```
Internet Monitor endpoint URL: https://internetmonitor.us-east-1.api.aws
```

Update an IAM policy for IPv6

IAM policies use the `aws:SourceIp` filter to set an allowed range of IP addresses.

Dual-stack supports both IPv4 and IPV6 traffic. If your network uses dual-stack, you must ensure that any IAM polices that are used for IP address filtering are updated to include IPv6 address ranges.

For example, this policy allows IPv4 address ranges `192.0.2.0.*` and `203.0.113.0.*`, identified in the `Condition` element.

```
# https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_examples_aws_deny-ip.html
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "aws:SourceIp": [
                    "192.0.2.0/24",
                    "203.0.113.0/24"
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
    }
}
```

To update this policy, change the policy's `Condition` element to add IPv6 address ranges, as shown in the following example:

```
"Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*", <<Existing IPv4 address - DO NOT REMOVE>>
                    "*203.0.113.0/24*", <<Existing IPv4 address  - DO NOT REMOVE>>
                    "*2001:DB8:1234:5678::/64*", <<New IPv6 IP address>>
                    "*2001:cdba:3257:8593::/64*" <<New IPv6 IP address>>
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
```

Important

Do not remove the existing IPv4 addresses in the policy. They are required for backward compatibility.

For more information about managing access permissions with IAM, see Managed policies and inline policies in the *AWS Identity and Access Management User Guide*.

Test the network after updating policies

After you update your IAM policies to include support for IPv6 addresses, we recommend that you test that your network can access an IPv6 endpoint. This section provides several examples, depending on the operating system that you use.

Test network with Linux/Unix or Mac OS X

If you use Linux/Unix or Mac OS X, you can test that your network can access the IPv6 endpoint by using the following curl command.

```
curl -v -s -o /dev/null http://ipv6.ec2-reachability.amazonaws.com/
```

If you are connected over IPv6, the connected IP address displays information similar to the following:

```
* About to connect() to aws.amazon.com port 443 (#0)
*   Trying IPv6 address... connected
* Connected to aws.amazon.com (IPv6 address) port 443 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1t
 zlib/1.2.3
> Host: aws.amazon.com
```

Test network with Windows

If you use Windows, you can test that your network can access a dual-stack endpoint over IPv6 or IPv4 by using a `ping` command, such as the following:

```
ping aws.amazon.com
```

If `ping` accesses the endpoint over IPv6, the command returns IPv6 addresses.

Verify that clients can support IPv6

We recommend that before you switch to using the *internetmonitor.{region}.api.aws* endpoint, that you first verify that your clients can access other AWS service endpoints that are already IPv6-enabled. The following steps describe how to verify this by using an existing IPv6 endpoint.

This example uses Linux and curl version 8.6.0, and uses the Amazon Athena service, which has IPv6-enabled endpoints located at the *api.aws* domain.

> **Note**
>
> Switch your AWS Region to the same Region where the client is located. In this example, we use the US East (N. Virginia) – `us-east-1` endpoint.

Use the following example to verify that your clients can access an IPv6-enabled AWS endpoint.

1. Verify that the Athena endpoint resolves with an IPv6 address by using the following command.

   ```
   dig +short AAAA athena.us-east-1.api.aws
   2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
   2600:1f18:e2f:4e03:4a1e:83b0:8823:4ce5
   ```

```
2600:1f18:e2f:4e04:34c3:6e9a:2b0d:dc79
```

2. Now, determine if your client network can make a connection using IPv6 by using the following command:

```
curl --ipv6 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws

remote ip: 2600:1f18:e2f:4e05:1a8a:948e:7c08:d2d6
response code: 404
```

If the remote IP address was identified **and** the response code is not `0`, a network connection was successfully made to the endpoint using IPv6.

If the remote IP address is blank or the response code is `0`, the client network or the network path to the endpoint is IPv4-only. You can verify this with the following curl command:

```
curl -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://athena.us-east-1.api.aws

remote ip: 3.210.103.49
response code: 404
```

If you run this command, and a remote IP address was identified **and** the response code is not `0`, a network connection was successfully made to the endpoint using IPv4.

How Internet Monitor works with IAM

Before you use IAM to manage access to Internet Monitor, learn what IAM features are available to use with Internet Monitor.

To see tables showing a similar high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

IAM feature	Internet Monitor support
Identity-based policies	Yes

IAM feature	Internet Monitor support
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for Internet Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Resource-based policies within Internet Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that

support resource-based policies, service administrators can use them to control access to a specific resource.

Policy actions for Internet Monitor

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Internet Monitor actions, see Actions defined by Internet Monitor in the *Service Authorization Reference*.

Policy actions in Internet Monitor use the following prefix before the action:

```
internetmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "internetmonitor:action1",
      "internetmonitor:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "internetmonitor:Describe*"
```

Policy resources for Internet Monitor

Supports policy resources: Yes

In the *Service Authorization Reference*, you can see the following information related to Internet Monitor:

- To see a list of Internet Monitor resource types and their ARNs, see Resources defined by Internet Monitor.
- To learn the actions that you can specify with the ARN of each resource, see Actions defined by Internet Monitor.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Policy condition keys for Internet Monitor

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Internet Monitor condition keys, see Condition keys for Internet Monitor in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Internet Monitor.

ACLs in Internet Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Internet Monitor

Supports ABAC (tags in policies): Partial

Internet Monitor has *partial* support for tags in policies. It supports tagging for one resource, monitors.

To use tags with Internet Monitor, use the AWS Command Line Interface or an AWS SDK. Tagging for Internet Monitor is not supported with the AWS Management Console.

To learn more about using tags in policies in general, review the following information.

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Internet Monitor

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Internet Monitor

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Internet Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

Service-linked role for Internet Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For more information about the service-linked role for Internet Monitor, see Service-linked role for Internet Monitor.

For details about creating or managing service-linked roles in general in AWS, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Cross-service confused deputy prevention

A confused deputy is an entity (a service or an account) that is coerced by a different entity to perform an action. This type of impersonation can happen cross-account and cross-service.

To prevent confused deputies, AWS provides tools that help you protect your data for all services using service principals that have been given access to resources in your AWS account. This section focuses on cross-service confused deputy prevention specific to Internet Monitor; however, you can learn more about this topic in the confused deputy problem section of the *IAM User Guide*.

To limit the permissions that IAM gives to Internet Monitor to access your resources, we recommend using the global condition context keys `aws:SourceArn` and `aws:SourceAccount` in your resource policies.

If you use both of these global condition context keys, and the `aws:SourceArn` value contains the AWS account ID, the `aws:SourceAccount` value and the AWS account in `aws:SourceArn` must use the same AWS account ID when used in the same policy statement.

For Internet Monitor, you specify your account ID for `aws:SourceAccount` and your monitor ARN for `aws:SourceArn`. For cross-service access, you also use your monitor ARN for `aws:SourceArn`.

> **Note**
>
> The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the **full ARN** of the resource. If you don't know the full ARN, or if you're specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:internetmonitor:us-east-1:111122223333:*`.

The following is an example of an assume role policy that shows how you can prevent a confused deputy issue.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "internetmonitor.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:internetmonitor:us-east-1:111122223333:monitor/
confused-deputy-monitor"
      },
      "StringEquals": {
        "aws:SourceAccount": "111122223333"
      }
    }
  }
```

```
}
```

AWS managed policies for Internet Monitor

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: CloudWatchInternetMonitorServiceRolePolicy

This policy is attached to the service-linked role named **AWSServiceRoleForInternetMonitor** to allow Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources or Network Load Balancers, so that you can select them when you create a monitor. For more information, see Service-linked role for Internet Monitor.

AWS managed policy: CloudWatchInternetMonitorReadOnlyAccess

You can attach `CloudWatchInternetMonitorReadOnlyAccess` to your IAM entities. This policy grants access to read-only actions to work with monitors and data in with Internet Monitor. Attach it to IAM users and other principals who need access to only read-only actions.

Specifically, the scope of this policy includes `internetmonitor:` so that users can use read-only Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch metrics. It includes some `logs:` policies to manage log queries.

To view the permissions for this policy, see CloudWatchInternetMonitorReadOnlyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: CloudWatchInternetMonitorFullAccess

You can attach `CloudWatchInternetMonitorFullAccess` to your IAM entities. This policy grants full access to Actions for Internet Monitor for working with Internet Monitor. Attach it to IAM users and other principals who need full access to Internet Monitor actions.

Specifically, scope of this policy includes `internetmonitor:` so that users can use Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `ec2:`, `cloudfront:`, `elasticloadbalancing:`, and `workspaces:` policies to work with resources that you add to monitors so that Internet Monitor can create a traffic profile for your application. It contains some `iam:` policies to manage IAM roles.

To view the permissions for this policy, see CloudWatchInternetMonitorFullAccess in the *AWS Managed Policy Reference*.

Internet Monitor updates to AWS managed policies

To view details about updates to AWS managed policies for Internet Monitor since this service began tracking these changes, see CloudWatch updates to AWS managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Service-linked role for Internet Monitor

Internet Monitor uses an AWS Identity and Access Management (IAM) service-linked role. A service-linked role is a unique type of IAM role that is linked directly to Internet Monitor. The service-linked role is predefined by Internet Monitor and includes all the permissions that the service requires to call other AWS services on your behalf.

Internet Monitor defines the permissions of the service-linked role, and unless defined otherwise, only Internet Monitor can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your Internet Monitor resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Internet Monitor

Internet Monitor uses the service-linked role named **AWSServiceRoleForInternetMonitor**. This role allows Internet Monitor to access resources in your account, such as Amazon Virtual Private Cloud resources, Amazon CloudFront distributions, Amazon WorkSpaces directories, and Network Load Balancers, so that you can select them when you create a monitor.

This service-linked role uses the managed policy `CloudWatchInternetMonitorServiceRolePolicy`.

The **AWSServiceRoleForInternetMonitor** service-linked role trusts the following service to assume the role:

- `internetmonitor.amazonaws.com`

To view the permissions for this policy, see CloudWatchInternetMonitorServiceRolePolicy in the *AWS Managed Policy Reference*.

Creating a service-linked role for Internet Monitor

You do not need to manually create the service-linked role for Internet Monitor. The first time that you create a monitor, Internet Monitor creates **AWSServiceRoleForInternetMonitor** for you.

For more information, see Creating a service-linked role in the *IAM User Guide*.

Editing a service-linked role for Internet Monitor

After Internet Monitor creates a service-linked role in your account, you cannot change the name of the role because various entities might reference the role. You can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Internet Monitor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete the role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for the service-linked role before you can manually delete it.

After you've removed your resources from your monitors in Internet Monitor and then deleted the monitors, you can delete the service-linked role **AWSServiceRoleForInternetMonitor**.

> **Note**
>
> If the Internet Monitor service is using the role when you try to delete it, then the deletion might fail. If that happens, wait for a few minutes and then try again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the **AWSServiceRoleForInternetMonitor** service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Updates to the Internet Monitor service-linked role

For updates to **AWSServiceRoleForInternetMonitor**, the AWS managed policy for the Internet Monitor service-linked role, see CloudWatch updates to AWS managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

Using Network Synthetic Monitor

Network Synthetic Monitor provides visibility into the performance of the network connecting your AWS hosted applications to your on-premises destinations, and allows you to identify the source of any network performance degradation within minutes. Network Synthetic Monitor is fully managed by AWS, and doesn't require separate agents on monitored resources. Use Network Synthetic Monitor to visualize packet loss and latency of your hybrid network connections, and set alerts and thresholds. Then, based on this information, you can take action to improve your end users' experience.

Network Synthetic Monitor is intended for network operators and application developers who want real-time insights into network performance.

Network Synthetic Monitor key features

- Use Network Synthetic Monitor to benchmark your changing hybrid network environment with continuous real-time packet loss and latency metrics.
- You can use the network health indicator (NHI) to rapidly diagnose whether network degradation is within the AWS network. The NHI applies to paths that reach a destination through AWS

Direct Connect, and to inter-Region paths that reach a destination in another Region. For more information about the supported path types, see Network health indicator for AWS. Network Synthetic Monitor writes the NHI to your Amazon CloudWatch account. The NHI metric is a binary value: 0 or 100. This value is based on a probabilistic score about whether network degradation is within AWS.

- Network Synthetic Monitor provides a fully-managed agent approach to monitoring, so you don't need to install agents either on VPCs or on-premises. To get started, you just need to specify a VPC subnet and an on-premises IP address. You can establish a private connection between your VPC and Network Synthetic Monitor resources by using AWS PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.
- Network Synthetic Monitor publishes metrics to CloudWatch Metrics. You can create dashboards to view your metrics, and also create actionable thresholds and alarms on the metrics that are specific to your application.

For more information, see How Network Synthetic Monitor works.

Network Synthetic Monitor terminology and components

- **Probes** — A probe is the traffic that's sent from an AWS-hosted resource to an on-premises destination IP address. Network Synthetic Monitor metrics measured by the probe are written into your CloudWatch account for every probe that's configured in a monitor.
- **Monitor** — A monitor displays network performance and other health information for traffic that you have created Network Synthetic Monitor *probes* for. You add probes as part of creating a monitor, and then you can view network performance metrics information using the monitor. When you create a monitor for an application, you add an AWS hosted resource as the network source. Network Synthetic Monitor then creates a list of all possible probes between the AWS hosted resource and your destination IP addresses. You select the destinations that you want to monitor traffic for.
- **AWS network source** — An AWS network source is a monitor probe's originating AWS source, which is a subnet in one of your VPCs.
- **Destination** — A destination is the target in your on-premises network for the AWS network source. A destination is a combination of your on-premises IP addresses, network protocols, ports, and network packet size. IPv4 and IPv6 addresses are both supported.

Network Synthetic Monitor requirements and limitations

The following summarizes requirements and limitations for Network Synthetic Monitor. For specific quotas (or limits), see Network Synthetic Monitor.

- Monitor subnets must be owned by the same account as the monitor.
- Network Synthetic Monitor doesn't provide automatic network failover in the event of an AWS network issue.
- There's a charge for each probe that you create. For pricing details, see Pricing for Network Synthetic Monitor.
- Don't use reserved or AWS-internal IP addresses as probe destinations. Network Synthetic Monitor can't correctly measure round-trip time and packet loss for these addresses because the probe traffic doesn't leave the AWS data center, which results in round-trip time values below 1 millisecond. Specifically, avoid the following destination addresses:
  - Any address in the link-local range `169.254.0.0/16`. This range includes the Instance Metadata Service address `169.254.169.254` and the addresses `169.254.254.1` and `169.254.254.2`, which are used by Network Synthetic Monitor probe infrastructure.
  - The address `10.0.0.2`.

How Network Synthetic Monitor works

Network Synthetic Monitor is fully managed by AWS, and doesn't require separate agents on monitored resources. Instead, you specify *probes* by providing a VPC subnet and on-premises IP addresses.

When you create a monitor in Network Synthetic Monitor for AWS-hosted resources, AWS creates and manages the infrastructure in the background that is required to perform round-trip time and packet loss measurements. Because AWS manages the required configurations, you can scale your monitoring rapidly, without needing to install or uninstall agents within your AWS infrastructure.

When probes are created, customized elastic network interfaces (ENIs) are created and attached to probe instances and customer subnets. If Network Synthetic Monitor replaces a probe instance, for example, if it becomes unhealthy, Network Synthetic Monitor detaches the ENIs and reattaches them to the probe replacement. This means that ENI IP addresses are not changed after they are created, unless you delete a probe and create a new one for the same source and destination.

Network Synthetic Monitor focuses monitoring on the routes taken by flows from your AWS-hosted resources instead of broadly monitoring all flows from your AWS Region. If your workloads spread across multiple Availability Zones, Network Synthetic Monitor can monitor routes from each of your private subnets.

Network Synthetic Monitor publishes round-trip time and packet loss metrics to your Amazon CloudWatch account, based on the aggregation interval that you set when you create a monitor. You can also use CloudWatch to set individual latency and packet loss thresholds for each monitor. For example, you might create an alarm for a packet loss sensitive workload to notify you if the packet loss average is higher than a static 0.1% threshold. You can also use CloudWatch anomaly detection to alarm on packet loss or latency metrics that are outside your desired ranges.

Availability and performance measurements

Network Synthetic Monitor sends periodic active probes from your AWS resource to your on-premises destinations. When you create a monitor, you specify the following:

- **Aggregation interval:** The time, in seconds, that CloudWatch receives the measured results. This will be either every 30 or 60 seconds. The aggregation period you choose for the monitor applies to all probes in that monitor.
- **Probe sources (AWS resources):** A source for a probe is a VPC and associated subnets, or just a VPC subnet, in the Regions where your network operates.
- **Probe destinations (customer resources):** A destination for a probe is the combination of on-premises IP addresses, network protocols, ports, and network packet size.
- **Probe protocol:** One of the supported protocols, ICMP or TCP. For more information, see Supported communication protocols.
- **Port (for TCP):** The port that your network uses to connect.
- **Packet size (for TCP):** The size, in bytes, of each packet transmitted between your AWS hosted resource and your destination on a single probe. You can specify a different packet size for each probe in a monitor.

A monitor publishes the following metrics:

- **Round-trip time:** This metric, measured in microseconds, is a measure of performance. It records the time it takes for the probe to be transmitted to the destination IP address and for the associated response to be received. The round-trip time is the average time observed during the aggregation interval.

- **Packet loss:** This metric measures the percentage of total packets sent and records the number of transmissions that didn't receive an associated response. No response implies that the packets were lost along the network path.

Probes on inter-Region paths send more traffic to cover the additional network paths. You can expect up to 240 packets per second (PPS) to each of these destinations.

Supported communication protocols

Network Synthetic Monitor supports two protocols for probes: ICMP and TCP.

ICMP-based probes carry ICMP echo requests from your AWS hosted resources to the destination address, and expect an ICMP echo reply in response. Network Synthetic Monitor uses the information on the ICMP echo request and reply messages to calculate round-trip time and packet loss metrics.

TCP-based probes carry TCP SYN packets from your AWS hosted resources to the destination address and port, and expect a TCP SYN+ACK packet in response. Network Synthetic Monitor uses the information on the TCP SYN and TCP SYN+ACK messages to calculate round-trip time and packet loss metrics. Network Synthetic Monitor periodically switches source TCP ports (using ports in the range 1024–65535) to increase network coverage, which increases the probability of detecting packet loss. Make sure that your firewall rules allow TCP traffic from this entire source port range to the configured destination port.

Network health indicator for AWS

Network Synthetic Monitor publishes a network health indicator (NHI) metric that provides information on issues with the AWS network for paths that include destinations connected through Direct Connect.

The NHI also supports inter-Region paths, which reach a destination in another AWS Region. The NHI supports the following inter-Region path types:

AWS PrivateLink

Paths to an interface VPC endpoint that connects to an endpoint service in another Region.

AWS Transit Gateway inter-Region peering

Paths that use a Transit Gateway peering attachment to reach another Region. For these paths, the NHI reflects the health of the AWS network path up to the Transit Gateway peering connection.

Amazon EC2 public IP address

Paths through an internet gateway to a public IP address of Amazon EC2 in another Region.

Inter-Region VPC peering

Paths that use a VPC peering connection to a VPC in another Region. A VPC peering connection within the same Region is not an inter-Region path.

For AWS PrivateLink, Amazon EC2 public IP address, and VPC peering paths, the NHI reflects the health of the AWS network path from the subnet where the monitor is deployed to the destination Region.

The NHI binary value is based on a statistical measure of the health of the AWS-controlled network path from the AWS hosted resource, where the monitor is deployed, to the Direct Connect location. Network Synthetic Monitor uses anomaly detection to calculate availability drops or lower performance along the network paths.

NHI is not accurate for Direct Connect attachments that use intermediary routing with Cloud WAN. When you have a hybrid network that includes Cloud WAN, do not use the NHI value as an indication of a performance issue.

Note

Each time that you create a new monitor, add a probe, or re-activate a probe, the NHI for the monitor is delayed by a few hours while AWS collects data to perform anomaly detection.

To provide the NHI value, Network Synthetic Monitor applies statistical correlation across AWS sample datasets, as well as to the packet loss and round-trip latency metrics for traffic simulating your network path. NHI can be one of two values: 100 or 0. A value of 100 indicates that Network Synthetic Monitor observed a network degradation within the AWS controlled network path. A value of 0 indicates that Network Synthetic Monitor did not observe any network degradation for

the AWS network along the path. Using the NHI value enables you to more quickly gain awareness of the cause of network issues. For example, you can set alerts on the NHI metric so you're notified about ongoing issues with the AWS network along your network paths.

Support for IPv4 and IPv6 addresses

Network Synthetic Monitor provides availability and performance metrics over IPv4 or IPv6 networks, and can monitor either IPv4 or IPv6 addresses from dual-stack VPCs. Network Synthetic Monitor doesn't allow both IPv4 and IPv6 destinations to be configured in the same monitor; you can create separate monitors for IPv4-only and IPv6-only destinations.

Supported AWS Regions for Network Synthetic Monitor

The AWS Regions where Network Synthetic Monitor is supported are listed in this section. For more information about Regions that Network Synthetic Monitor is supported in, including opt-in Regions, see Network Synthetic Monitor endpoints and quotas in the *Amazon Web Services General Reference*.

Region name	Region
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2

Region name	Region
Asia Pacific (Thailand)	ap-southeast-7
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Canada West (Calgary)	ca-west-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Milan)	eu-south-1
Europe (Paris)	eu-west-3
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1
Europe (Zurich)	eu-central-2
Israel (Tel Aviv)	il-central-1
Mexico (Central)	mx-central-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1
South America (São Paulo)	sa-east-1
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1

Region name	Region
US West (Oregon)	us-west-2

Pricing for Network Synthetic Monitor

With Network Synthetic Monitor, there are no upfront costs or long-term commitments. Pricing for Network Synthetic Monitor has the following two components:

- An hourly fee per monitored AWS resource
- CloudWatch metrics fees

When you create a monitor in Network Synthetic Monitor, you associate AWS resources (sources) with it to be monitored. For Network Synthetic Monitor, these resources are subnets in Amazon Virtual Private Cloud (VPC). For each resource, you can create up to four probes, each of which is for traffic from a subnet in the VPC to four of your destination IP addresses. To help control your bill, you can adjust your subnet coverage and on-premises IP address destination coverage by reducing the number of resources that you monitor.

For more information about pricing, see the Amazon CloudWatch pricing page.

Network Synthetic Monitor API operations

The following table lists Network Synthetic Monitor API operations that you can use with Amazon CloudWatch. Refer to this table for links to relevant documentation.

Action	API reference	More information
Create a monitor between a source subnet and destination IP address.	See CreateMonitor	See Create a monitor
Create a probe within a monitor.	See CreateProbe	See Activate or deactivate a probe
Remove a monitor.	See DeleteMonitor	See Delete a monitor

Action	API reference	More information
Delete a specific probe.	See DeleteProbe	See Delete a probe
Get information about a monitor.	See GetMonitor	See Working with monitors and probes in Network Synthetic Monitor
Get information about a specific probe.	See GetProbe	See Working with monitors and probes in Network Synthetic Monitor
Get a list of all your monitors.	See ListMonitors	See Working with monitors and probes in Network Synthetic Monitor
List tags assigned to a resource.	See ListTagsForResource	See Tag or untag resources
Add key-value pairs, or tags, to a monitor or probe.	See TagResource	See Tag or untag resources
Remove a key-value pair, or tag, from a monitor or probe.	See UntagResource	See Tag or untag resources
Update an aggregation period for a monitor.	See UpdateMonitor	See Edit a monitor
Update a probe in a monitor.	See UpdateProbe	See Edit a probe

Working with monitors and probes in Network Synthetic Monitor

To get started, create a monitor with probes in Network Synthetic Monitor to measure network performance over a specified aggregation period. Then, you can update a monitor to make desired changes, for example, to change the aggregation period, deactivate or activate probes, or add or remove tags.

The following sections provide step-by-step instructions for completing these tasks for your monitors and probes by using the Amazon CloudWatch console. You can also make changes to your monitor by using the AWS Command Line Interface.

Topics

- Create a monitor
- Edit a monitor
- Delete a monitor
- Activate or deactivate a probe
- Add a probe to a monitor
- Edit a probe
- Delete a probe
- Tag or untag resources

Create a monitor

The following sections describe how to create a monitor in Network Synthetic Monitor, including the required probes. When you create a monitor, you specify probes by choosing source subnets, and then adding up to four destinations for each one. Each source-destination pair is a probe.

You can make changes to a monitor after you create it, for example, to add, remove, or deactivate probes. For more information, see Working with monitors and probes in Network Synthetic Monitor.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

The following procedures provide step-by-step instructions for how to create a monitor by using the Amazon CloudWatch console.

- Define monitor details
- Choose sources and destinations
- Confirm probes
- Review and create monitor

> **Important**
>
> These steps are designed to be completed all at once. You can't save in-process work to continue later.

Define monitor details

The first step to create a monitor is to define the basic details, by giving the monitor a name and defining the aggregation period. Optionally, you can also add tags.

To define monitor details

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. Choose **Create monitor**.
3. For **Monitor name**, enter a name for the monitor.
4. For **Aggregation period**, choose how often you want to send metrics to CloudWatch: **30 seconds** or **60 seconds**.

   > **Note**
   >
   > A shorter aggregation period provides faster detection of network issues. However, the aggregation period that you choose can affect your billing costs. For more information about pricing, see the Amazon CloudWatch pricing page.

5. (Optional) For **Tags**, add **Key** and **Value** pairs to help identify this resource, so that you can search or filter on specific information.

   1. Choose **Add new tag**.
   2. Enter a **Key** name and associated **Value**.
   3. Choose **Add new tag** to add the new tag.

      You can add multiple tags by choosing **Add new tag**, or you can remove a tag by choosing **Remove**.
   4. If you want to associate your tags with the probes for the monitor, keep **Add tags to probes created by monitor** selected. This adds the tags to the monitor probes, which can be helpful for tag-based authentication or metering.

6. Choose **Next**. On the next page, you'll specify sources and destinations, to create probes for the monitor.

Choose sources and destinations

For each monitor in Network Synthetic Monitor, you specify one or more probes, which are a combination of an AWS source and a destination.

- A source for a probe is a VPC and associated subnets (or just a VPC subnet) in the Regions where your network operates.
- A destination is the combination of on-premises IP addresses, network protocols, ports, and network packet size.

> **Important**
>
> These steps are designed to be completed all at once. You can't save in-process work to continue later.

To choose sources and destinations

1. Prerequisite: Define monitor details.
2. Under **AWS network source**, choose one or more subnets to include in the monitor. To choose all subnets within a VPC, choose the VPC. Or, choose specific subnets within a VPC. The subnets that you choose are the monitor sources.
3. For **Destination 1**, enter a destination IP address of the on-premises network. IPv4 and IPv6 addresses are both supported.

   > **Note**
   >
   > Don't enter a reserved or AWS-internal IP address as a destination, such as an address in the link-local range `169.254.0.0/16` or the address `10.0.0.2`. Network Synthetic Monitor can't correctly measure round-trip time and packet loss for these addresses, and reports round-trip time values below 1 millisecond. For more information, see Network Synthetic Monitor requirements and limitations.

4. Choose **Advanced settings**.

5. For **Protocol**, choose the network protocol for the on-premises destination. The protocol can be either **ICMP** or **TCP**.
6. If you choose **TCP**, enter the following information:
    1. Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
    2. Enter the **Packet size**. This is the size, in bytes, of each packet that's sent on the probe, between the source and destination. Packet size must be a number from **56** to **8500**.
7. Choose **Add destination** to add another on-premises destination to the monitor. Repeat these steps for each destination that you want to add.
8. When you're finished adding sources and destinations, choose **Next** to confirm the probes for the monitor.

Confirm probes

On the **Confirm probes** page, review all the probes that will be created for the monitor, to make sure that they're the correct set of sources and destinations.

The **Confirm probes** page shows all the possible combinations of the sources and destinations for the probe specifications that you provided in the previous step. For example, if you have six source subnets and four destination IP addresses, there are 24 possible probe combinations, so 24 probes will be created.

> **Important**
>
> - These steps are intended to be completed in one session. You can't save in-process work to continue later.
> - The **Confirm probes** page does not indicate whether a probe is valid. We recommend that you review this page carefully, and then delete any probes that aren't valid. You might be charged for probes that aren't valid if you don't remove them.

To confirm monitor probes

1. Prerequisite: Choose sources and destinations.
2. On the **Confirm probes** page, review the list of source and destination probe combinations.

3. Choose any probes that you want to remove from the monitor, and then choose **Remove**.

> **Note**
>
> You are not prompted to confirm deleting a probe. If you delete a probe and want to restore it, you must set it up again. You can add a probe to an existing monitor by following the steps in Add a probe to a monitor.

4. Choose **Next**, and then review the monitor details.

Review and create monitor

The final step is to review the details of the monitor and the probes for the monitor, and then create the monitor. You can change any information about the monitor at this point.

When you have finished reviewing and updating any information that isn't correct, create the monitor.

As soon as you create the monitor, Network Synthetic Monitor begins tracking metrics and you'll start being charged for probes in the monitor.

> **Important**
>
> - This step is intended to be completed in one session. You can't save in-process work to continue later.
> - If you choose to edit a section, you must step through the process to create a monitor from the point that you make the edits. Earlier monitor creation pages maintain the information that you already entered.

To review and create a monitor

1. On the **Review and create probes** page, choose **Edit** for any section where you want to make changes.
2. Make changes in that section, and then choose **Next**.
3. When you're finished making edits, choose **Create monitor**.

The Network Synthetic Monitor page displays the current state of monitor creation in the **Monitors** section. When Network Synthetic Monitor is still creating the monitor, the **State** is **Pending**. When the **State** changes to **Active**, you can view CloudWatch metrics in the monitor dashboard.

For information on working with the monitor dashboard, see Network Synthetic Monitor dashboards.

> **Note**
>
> It can take several minutes for a newly-added monitor to begin collecting network metrics.

Edit a monitor

You can edit information for a Network Synthetic Monitor, including change the name, setting a new aggregation period, or adding or removing tags. Changing a monitor's information does not change any of its associated probes.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To edit a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the monitor that you want to edit.
3. On the monitor dashboard page, choose **Edit**.
4. For the **Monitor name**, enter the new name for the monitor.
5. For the **Aggregation period**, choose how often you want to send metrics to CloudWatch. Valid periods are:
   - **30 seconds**
   - **60 seconds**

> **Note**
>
> A shorter aggregation period provides faster detection of network issues. However, the aggregation period that you choose can affect your billing costs. For more information about pricing, see the Amazon CloudWatch pricing page.

6. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information. You can also just change the **Value** of any current **Key**.

   1. Choose **Add new tag**.
   2. Enter a **Key** name and associated **Value**.
   3. Choose **Add new tag** to add the new tag.

      You can add multiple tags by choosing **Add new tag**, or you can remove a tag by choosing **Remove**.
   4. If you want to associate your tags with the monitor, keep **Add tags to probes created by monitor** checked. This adds the tags to the monitor probes, which can be helpful if you're using tag-based authentication or metering.

7. Choose **Save changes**.

Delete a monitor

Before you can delete a monitor in Network Synthetic Monitor, you must deactivate or delete all probes associated with the monitor, regardless of the monitor **State**. After a monitor is deleted, you are no longer be charged for probes in the monitor. Be aware that you can't restore a deleted monitor.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To delete a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the monitor that you want to delete.
3. Choose **Actions**, and then choose **Delete**.
4. If you have any active probes for the monitor, you're prompted to deactivate them. Choose **Deactivate probes**.

   > **Note**
   >
   > You can't cancel or undo this action after you choose **Deactivate probes**. Deactivated probes, however, aren't removed from the monitor. If you like, you can reactivate them later. For more information, see Activate or deactivate a probe.

5. Enter **`confirm`** in the confirmation field, and then choose **Delete**.

Alternatively, you can delete a monitor programmatically, for example, by using the AWS Command Line Interface.

To delete a monitor by using the CLI

1. To delete a monitor, you need the monitor name. If you don't know the name, get a list of your monitors by running the list-monitors command. Note the name of the monitor that you want to delete.
2. Verify whether the monitor contains any active probes. Use get-monitor with the monitor name from the previous step. This returns a list of any probes associated with that monitor.
3. If the monitor contains active probes, you must first either set the probes to inactive or delete them.
   - To set a probe to inactive, use update-probe, and set the state to `INACTIVE`.
   - To delete a probe, use delete-probe.
4. After the probes are set to `INACTIVE` or deleted, you can delete the monitor by running the delete-monitor command. Inactive probes are not deleted when you delete the monitor.

Activate or deactivate a probe

You can activate or deactivate a probe in a monitor in Network Synthetic Monitor. You might want to deactivate a probe, for example, if you aren't currently using it but might want to use it again in the future. By deactivating a probe instead of deleting it, you won't need to spend time setting it up again. You are not billed for deactivated probes.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To set a probe to active or inactive by using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the probe that you want to activate or deactivate.
4. Choose **Actions**, and then choose **Activate** or **Deactivate**.

> **Note**
>
> When you reactivate a probe, you begin incurring billing charges on the probe again.

Add a probe to a monitor

You can add a probe to an existing monitor in Network Synthetic Monitor. Note that when you add probes to a monitor, your billing structure is updated to include the new probe.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To add a probe to a monitor using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.

2. In the **Monitors** section, do one of the following:

   - Choose the **Name** link of the monitor that you want to add a probe to. Choose the **Monitor details** tab, and then in the **Probes** section, choose **Add probe**.
   - Choose the monitor check box, choose **Actions**, and then choose **Add probe**.

3. On the **Add probe** page, do the following:

   1. Under **AWS network source**, choose a subnet to add to the monitor.

      > **Note**
      >
      > You can only add one probe at a time and up to four probes per monitor.

   2. Enter the destination **IP address** of the on-premises network. Both IPv4 and IPv6 addresses are supported.
   3. Choose **Advanced settings**.
   4. Choose the network **Protocol** for the destination. It can be either **ICMP** or **TCP**.
   5. If the **Protocol** is **TCP**, enter the following information. Otherwise, skip to the next step:
      - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
      - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.

4. (Optional) In the **Tags** section, add **Key** and **Value** pairs to further help identify this resource, allowing you to search or filter on specific information.

   1. Choose **Add new tag**.
   2. Enter a **Key** name and associated **Value**.
   3. Choose **Add new tag** to add the new tag.

   You can add multiple tags by choosing **Add new tag**, or you can remove any tag by choosing **Remove**.

5. Choose **Add probe**.

While the probe is being activated, the **State** shows **Pending**. It might take several minutes for the probe to become **Active**.

Edit a probe

You can change any information for an existing probe, regardless of whether that probe is active or inactive.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To edit a probe by using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.

   Under **Name**, choose a monitor link to open the monitor dashboard.
2. Choose the **Monitor details** tab.
3. In the **Probes** section, choose the link for the probe that you want to edit.
4. On the probe details page, choose **Edit**.
5. On the **Edit *probe*** page, enter the new destination **IP address** for the probe. IPv4 and IPv6 addresses are both supported.
6. Choose **Advanced settings**.
7. Choose a network **Protocol**, **ICMP** or **TCP**.
8. If the **Protocol** is **TCP**, enter the following information:

   - Enter the **Port** that your network uses to connect. The port must be a number from **1** to **65535**.
   - Enter the **Packet size**. This is the size, in bytes, of each packet sent along the probe between the source and destination. Packet size must be a number from **56** to **8500**.
9. (Optional) Add, change, or remove Tags for the probe.
10. Choose **Save changes**.

Delete a probe

You can delete a probe rather than deactivating it if you know that you won't need it again later. You can't recover a deleted probe; instead, you must recreate it. Billing charges end for a probe when the probe is deleted.

You can work with monitors and probes by using either the Amazon CloudWatch console or the AWS Command Line Interface. To work with Network Synthetic Monitor programmatically, see the Network Synthetic Monitor API Reference and networkmonitor in the AWS Command Line Interface Command Reference.

To delete a probe using the console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, under **Name**, choose a monitor link to open the monitor dashboard.
3. Choose the **Monitor details** tab.
4. Choose the monitor check box, choose **Actions**, and then choose **Delete**.
5. In the **Delete probe** dialog box, do the following:
6. Choose **Delete** to confirm that you want to delete the probe.

   The **State** of the probe in the **Probes** section shows **Deleting**. After it's deleted, the probe is removed from the **Probes** section.

Tag or untag resources

You can work with resource tags in Network Synthetic Monitor, to add or remove tags.

You can update tags by updating monitors or probes in the console. Or, you can work with tags programmatically, for example, by using the AWS Command Line Interface.

To update monitor tags by using the CLI

- To list resource tags, use list-tags-for-resources.
- To tag a resource, use tag-resource.
- To untag a resource, use untag-resource.

Network Synthetic Monitor dashboards

You can use dashboards in Network Synthetic Monitor to determine if a network issue is caused by AWS, by using the network health indicator (NHI), and view probe round-trip time and packet loss. You can view this information and metrics for monitors, as well as for individual probes.

Network Synthetic Monitor creates several metrics that you can view in CloudWatch Metrics. You can specify alarms for the metrics that Network Synthetic Monitor returns. For more information, see Probe alarms.

Topics

- Monitor dashboards
- Probe dashboards
- Specify metrics time frame

Monitor dashboards

You can use the monitor dashboard in Network Synthetic Monitor to view the network health indicator (NHI), as well as probe round-trip time and packet loss at a monitor level. That is, a monitor dashboard shows this information for all the probes created for the monitor.

Network Synthetic Monitor also has dashboards for probes, to view the information at a probe level. For more information, see Probe dashboards.

To access a monitor dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the **Name** link to open the monitor dashboard.

Overview page

The **Overview** page displays the following information for a monitor:

- **Network health** — Network health displays the network health indicator (NHI) value, which pertains to health of only the AWS network. The NHI status is displayed as **Healthy** or **Degraded**. A **Healthy** status indicates that Network Synthetic Monitor did not observe issues with the AWS network. A **Degraded** status indicates that Network Synthetic Monitor observed an issue with the AWS network. The status bar in this section shows the status of the network health indicator over a default time of one hour. Hover over any point in the status bar to view additional details.
- **Probe traffic summary** — Displays the current state of the traffic between the source AWS subnets specified for the probes in the monitor and the probes' destination IP addresses. This summary displays the following:

- **Probes in alarm** — This number indicates how many of your probes in this monitor are in a degraded state. An alarm is triggered when a metric that you've set up as an alarm is triggered. For information on creating alarms for metrics in Network Synthetic Monitor, see Probe alarms.
- **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address. This is represented as a percentage of the total packets sent.
- **Round-trip time** — The time it takes, displayed in milliseconds, for a packet from the source subnet to reach the destination IP address, and then come back again. Round-trip time is the average RTT observed during the aggregation period.

The data is represented on an interactive graph, so you can explore to learn details.

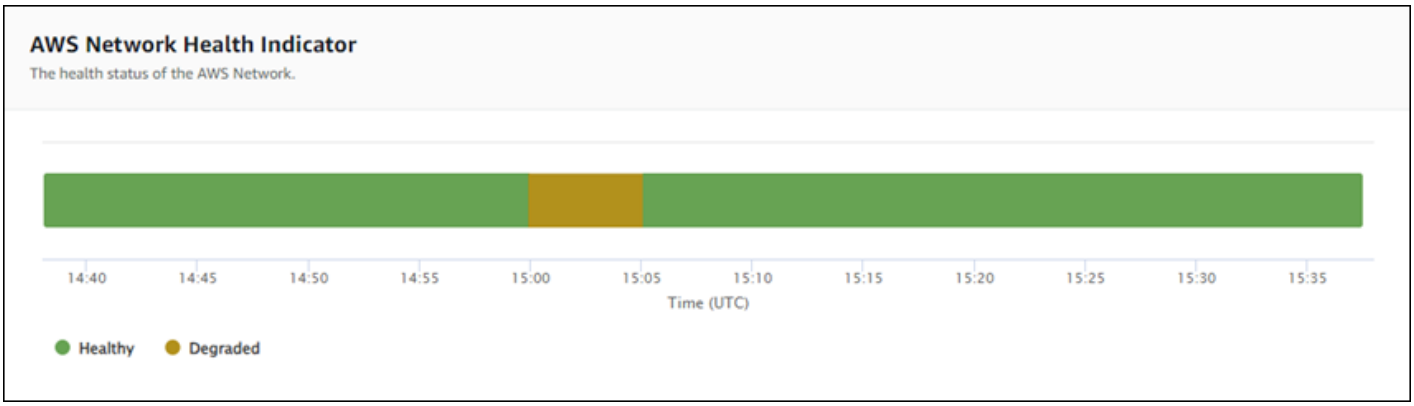
By default, data is displayed for a two-hour time frame, calculated from the current date and time. However, you can change the range to fit your needs. For more information, see Specify metrics time frame.

Tracking metrics

The **Overview** dashboard in Network Synthetic Monitor displays a graphical representation of a monitor and probes. The following graphs are shown:

- **Network health indicator** — This represents the the NHI values over a specified period. NHI indicates whether a network issue is due to problems with the AWS network. NHI is displayed as **Healthy** (no issue with the AWS network) or **Degraded** (there is an issue with the AWS network).

  In the following example, you can see that from 15:00 UTC until 15:05 UTC, there was a network issue that was due to an AWS network issue (**Degraded**). After 15:05, the network issue with the AWS network ended, so the value returned to **Healthy**. You can hover over any section of the graph to see additional details.



> **Note**
>
> The NHI indicates that an issue is due to the AWS network. It does not describe the overall health of the AWS network nor the health of Network Synthetic Monitor probes.

- **Packet loss** — This graph displays a line that shows the percentage of packet loss for each probe in a monitor. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. You can hover over a probe in the chart to see the source subnet, the destination IP address, and the percentage of packet loss.

  In the following example, a packet loss alarm was created for a probe from a subnet to IP address 127.0.0.1. The alarm was triggered when the packet loss threshold was exceeded for the probe. If you hover over the graph, you can see the probe source and destination, and that there was a 30.97% packet loss for this probe on November 21 at 02:41:30.



- **Round-trip time** — This graph displays a line that shows the round-trip time for each probe. The legend at the bottom of the page displays each of the probes in the monitor, color-coded for uniqueness. You can hover over a probe in the chart to see the source subnet, the destination IP address, and the round-trip time.

The following example shows that on Tuesday, Nov 21, at 21:45:30, the round-trip time for a probe from a subnet to IP address 127.0.0.1 was 0.075 seconds.



Monitor details

The **Monitor details** page displays details about your monitor, including a list of the probes for the monitor. You can update or add tags, or add a probe. The page includes the following sections:

- **Monitor details** — This page provides details about your monitor. You can't edit information in this section. However, you can view details of the Network Synthetic Monitor service-linked role: choose the **Role name** link to see details.
- **Probes** — This section displays a list of all probes associated with the monitor. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe, to activate or deactivate it. For more information, see Working with monitors and probes in Network Synthetic Monitor.

  The **Probes** section displays information about each probe set up for that monitor, including the probe **ID**, the **VPC ID**, the **Subnet ID**, **IP address**, **Protocol**, and whether the probe is **Active** or **Inactive**.

  If you've created an alarm for a probe, the current **Status** of the alarm is shown. A status of **OK** indicates that there are no metrics events have triggered any alarms. A status of **In alarm** indicates that a metric that you created in CloudWatch triggered an alarm. If no status is displayed for a probe, then there isn't a CloudWatch alarm for it. For information on the types of Network Synthetic Monitor probe alarms that you can create, see Probe alarms.

- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see Edit a monitor.

Probe dashboards

You can use a **Probe** dashboard in Network Synthetic Monitor to view the network health indicator (NHI) for a probe, as well as information about round-trip time and packet loss for specific probes. There are two dashboards for probes: an **Overview** page and **Probe details** page.

You can create CloudWatch alarms to set packet loss and round-trip time metric thresholds. When a threshold is reached for a metric, a CloudWatch alarm notifies you. For information on creating probe alarms, see Probe alarms.

To access a probe dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/, and then, under **Network Monitoring**, choose **Synthetic monitors**.
2. In the **Monitors** section, choose the **Name** link to open the dashboard for a specific monitor.
3. To view the dashboard for a specific probe, choose the **ID** link for the probe.

Overview page

The **Overview** page displays the following information for a probe:

- **Network health** — Network health displays the network health indicator (NHI) value, which pertains only to health of the AWS network. The NHI status is provided as **Healthy** or **Degraded**. A **Healthy** status indicates that Network Synthetic Monitor did not observe issues with the AWS network for a probe. A **Degraded** status indicates that Network Synthetic Monitor observed an issue with the AWS network. The status bar in this section shows the status of the network health indicator over a default time of one hour. Hover over any point in the status bar to view additional details.
- **Packet loss** — The number of packets that were lost from the source subnet to the destination IP address for this probe.
- **Round-trip time** — The time it takes, in milliseconds, for a packet from the source subnet to reach the destination IP address, and then come back again. Round-trip time (RTT) is the average RTT observed during the aggregation period.

Probe details

The **Probe details** page displays information about a probe, including the source and destination. You can also edit the probe, for example, to activate or deactivate it. For more information, see Working with monitors and probes in Network Synthetic Monitor.

- **Probe details** — This section provides general information about the probe, which can't be edited.
- **Probe source and destination** — This section displays details about the probe. Choose a **VPC** or **Subnet ID** link to open the VPC or subnet details in the Amazon VPC Console. You can modify a probe, for example, to activate or deactivate it.
- **Tags** — View the current tags for a monitor. You can add or remove tags by choosing **Manage tags**. This opens the **Edit probe** page. For more information on editing tags, see Edit a probe.

Specify metrics time frame

Metrics and events on the dashboards in Network Synthetic Monitor use a default time of two hours, calculated from the current time, but you can set a custom metrics default time frame to use. You can change the default to one of the following presets for the metrics time frame:

- **1h** — one hour
- **2h** — two hours
- **1d** — one day
- **1w** — one week

You can also set a custom time frame. Choose **Custom**, choose an **Absolute** or **Relative** time, and then set the time frame to a time of your own choosing. Relative time supports only 15 days back from today's date, following CloudWatch guidelines.

Additionally, you can choose the time displayed in the charts to be based on either the UTC time zone or a local time zone.

For more information, see Changing the time range or time zone format of a CloudWatch dashboard.

Probe alarms

You can create Amazon CloudWatch alarms based on Network Synthetic Monitor metrics, just as you can for other Amazon CloudWatch metrics. Any alarm that you create will appear in the probe's **Status** column of the **Monitor details** section of the Network Synthetic Monitor dashboard when the alarm is triggered. The status will either be **OK** or **In Alarm**. If no status displays for a probe, then no alarm was created for that probe.

For example, you can create an alarm based on the Network Synthetic Monitor metric `PacketLoss`, and configure it to send a notification when the metric is higher than a value that you choose. You configure alarms for Network Synthetic Monitor metrics following the same guidelines as for other CloudWatch metrics.

The following metrics are available under `AWS/NetworkMonitor` when creating a CloudWatch alarm for Network Synthetic Monitor.

- **HealthIndicator**
- **PacketLoss**
- **RTT (Round-trip time)**

For the steps to create a Network Synthetic Monitor alarm in CloudWatch, see Create a CloudWatch alarm based on a static threshold.

Data security and data protection in Network Synthetic Monitor

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Network Synthetic Monitor, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Network Synthetic Monitor. The following topics show you how to configure Network Synthetic Monitor to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Network Synthetic Monitor resources.

Topics

- Data protection in Network Synthetic Monitor
- Infrastructure Security in Network Synthetic Monitor

Data protection in Network Synthetic Monitor

The AWS shared responsibility model applies to data protection in Network Synthetic Monitor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.

- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Network Synthetic Monitor or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Infrastructure Security in Network Synthetic Monitor

As a managed service, Network Synthetic Monitor is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Network Synthetic Monitor through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Identity and access management for Network Synthetic Monitor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use Network Synthetic Monitor resources. IAM is an AWS service that you can use with no additional charge. You can use features of IAM to allow other users, services, and applications to use your AWS resources fully or in a limited way, without sharing your security credentials.

By default, IAM users don't have permission to create, view, or modify AWS resources. To allow an IAM user to access resources, such as a global network, and perform tasks, you must:

- Create an IAM policy that grants the user permission to use the specific resources and API actions they need
- Attach the policy to the IAM user or to the group to which the user belongs

When you attach a policy to a user or group of users, it allows or denies the user permissions to perform the specified tasks on the specified resources.

Condition keys

The `Condition` element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. For more information, see IAM JSON policy elements: Condition operators in the *AWS Identity and Access Management User Guide*.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name.

You can attach tags to Network Synthetic Monitor resources or pass tags in a request to Cloud WAN. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. See IAM JSON policy elements: Condition in the *AWS Identity and Access Management User Guide* for more information.

To see all AWS global condition keys, see AWS global condition context keys in the *AWS Identity and Access Management User Guide*.

Tag core network resources

A tag is a metadata label that either you or AWS assigns to an AWS resource. Each tag consists of a key and a value. For tags that you assign, you define the key and the value. For example, you might define the key as `purpose` and the value as `test` for one resource. Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.

- Control access to your AWS resources. For more information, see Controlling access to AWS resources using tags in the *AWS Identify and Access Management User Guide*.

How Network Synthetic Monitor works with IAM

Before you use IAM to manage access to Network Synthetic Monitor, learn what IAM features are available to use with Network Synthetic Monitor.

The following table shows the IAM features you can use with this service.

IAM feature	Network Synthetic Monitor support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how Network Synthetic Monitor and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Network Synthetic Monitor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Network Synthetic Monitor

To view examples of Network Synthetic Monitor identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Resource-based policies within Network Synthetic Monitor

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Network Synthetic Monitor

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Network Synthetic Monitor actions, see Actions defined by Network Synthetic Monitor in the *Service Authorization Reference*.

Policy actions in Network Synthetic Monitor use the following prefix before the action:

```
networkmonitor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "networkmonitor:action1",
      "networkmonitor:action2"
         ]
```

To view examples of Network Synthetic Monitor identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy resources for Network Synthetic Monitor

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Network Synthetic Monitor resource types and their ARNs, see Resources defined by Network Synthetic Monitor in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Network Synthetic Monitor.

Policy condition keys for Network Synthetic Monitor

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Network Synthetic Monitor condition keys, see Condition keys for Network Synthetic Monitor in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Network Synthetic Monitor.

ACLs in Network Synthetic Monitor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Network Synthetic Monitor

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Network Synthetic Monitor

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Network Synthetic Monitor

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Network Synthetic Monitor

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Network Synthetic Monitor functionality. Edit service roles only when Network Synthetic Monitor provides guidance to do so.

Using a service-linked role for Network Synthetic Monitor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Network Synthetic Monitor

By default, users and roles don't have permission to create or modify Network Synthetic Monitor resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Network Synthetic Monitor, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Network Synthetic Monitor in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Network Synthetic Monitor console
- Allow users to view their own permissions
- Troubleshooting Network Synthetic Monitor identity and access

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Network Synthetic Monitor resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Network Synthetic Monitor console

To access the Network Synthetic Monitor console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Network Synthetic Monitor resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To make sure that users and roles can still use the Network Synthetic Monitor console, also attach the Network Synthetic Monitor `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Network Synthetic Monitor identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Network Synthetic Monitor and IAM.

Topics

- I am not authorized to perform an action in Network Synthetic Monitor
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Network Synthetic Monitor resources

I am not authorized to perform an action in Network Synthetic Monitor

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `networkmonitor:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 networkmonitor:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `networkmonitor:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Network Synthetic Monitor.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Network Synthetic Monitor. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Network Synthetic Monitor resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Network Synthetic Monitor supports these features, see How Amazon CloudWatch works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

AWS managed policies for Network Synthetic Monitor

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: CloudWatchNetworkMonitorServiceRolePolicy

The `CloudWatchNetworkMonitorServiceRolePolicy` is attached to a service-linked role that allows the service to perform actions on your behalf and access resources associated with CloudWatch Network Synthetic Monitor. You cannot attach this policy to your IAM identities. For more information, see Using a service-linked role for Network Synthetic Monitor.

Network Synthetic Monitor updates to AWS managed policies

To view details about updates to AWS managed policies for Network Synthetic Monitor since this service began tracking these changes, see CloudWatch updates to AWS managed policies. For automatic alerts about managed policy changes in CloudWatch, subscribe to the RSS feed on the CloudWatch Document history page.

IAM permissions for Network Synthetic Monitor

To use Network Synthetic Monitor users must have the correct permissions.

For more information about security in Amazon CloudWatch, see Identity and access management for Amazon CloudWatch.

Permissions required to view a monitor

To view a monitor for Network Synthetic Monitor in the AWS Management Console, you must be signed in as a user or role that has the following permissions:

JSON

```
{
"Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "networkmonitor:Get*",
                "networkmonitor:List*"
                ],
            "Resource": "*"
        }
    ]
}
```

Permissions required to create a monitor

To create a monitor in Network Synthetic Monitor, users must have permission to create a service-linked role that is associated with Network Synthetic Monitor. To learn more about the service-linked role, see Using a service-linked role for Network Synthetic Monitor.

To create a monitor for Network Synthetic Monitor in the AWS Management Console, you must be signed in as a user or role that has the permissions included in the following policy.

> **Note**
>
> If you create an identity-based permissions policy that is more restrictive, users with that policy won't be able to create a monitor.

JSON

```
{
"Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "networkmonitor:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "networkmonitor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:GetRole",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
networkmonitor.amazonaws.com/AWSServiceRoleForNetworkMonitor"
        },
        {
            "Action": [
                "ec2:CreateSecurityGroup",
                "ec2:CreateNetworkInterface",
                "ec2:CreateTags"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Using a service-linked role for Network Synthetic Monitor

Network Synthetic Monitor uses the following service-linked role for the permissions that it requires to call other AWS services on your behalf:

- AWSServiceRoleForNetworkMonitor

AWSServiceRoleForNetworkMonitor

Network Synthetic Monitor uses the service-linked role named AWSServiceRoleForNetworkMonitor to update and manage monitors.

The AWSServiceRoleForNetworkMonitor service-linked role trusts the following service to assume the role:

- networkmonitor.amazonaws.com

The CloudWatchNetworkMonitorServiceRolePolicy is attached to the service linked role and grants access for the service to access VPC and EC2 resources in your account, as well as manage the monitors that you create.

Permissions groupings

The policy is grouped into the following sets of permissions:

- **cloudwatch** - This allows the service principal to publish network monitoring metrics to CloudWatch resources.
- **ec2** - This allows the service principal to describe VPCs and subnets in your account to create or update monitors and probes. The service principal can also describe AWS Transit Gateway peering attachments, VPC peering connections, and VPC endpoints to resolve the destination Region for a probe. It can create, modify, and delete security groups, network interfaces, and their associated permissions. This configures the monitor or probe to send monitoring traffic to your endpoints.

To view the permissions for this policy, see CloudWatchNetworkMonitorServiceRolePolicy in the *AWS Managed Policy Reference*.

Create the service-linked role

AWSServiceRoleForNetworkMonitor

You don't need to manually create the AWSServiceRoleForNetworkMonitor role.

- Network Synthetic Monitor creates the `AWSServiceRoleForNetworkMonitor` role when you create your first monitor with the feature. This role then applies to all additional monitors that you create.

To create a service-linked role on your behalf, you must have the required permissions. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Edit the service-linked role

You can edit the `AWSServiceRoleForNetworkMonitor` descriptions using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Delete the service-linked role

If you no longer need to use Network Synthetic Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your monitors. For more information, see Delete a monitor.

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` Network Synthetic Monitor will create the role again when you create a new monitor.

Supported Regions for the Network Synthetic Monitor service-linked role

Network Synthetic Monitor supports the service-linked role in all of AWS Regions where the service is available. For more information, see AWS endpoints in the *AWS General Reference*.

Delete the service-linked role

If you no longer need to use Network Synthetic Monitor, we recommend that you delete the `AWSServiceRoleForNetworkMonitor` role.

You can delete these service-linked roles only after you delete your monitors. For more information, see Delete a monitor.

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

After you delete `AWSServiceRoleForNetworkMonitor` Network Synthetic Monitor will create the role again when you create a new monitor.

CloudWatch observability solutions

CloudWatch observability solutions offer a catalog of readily available configurations to help you quickly implement monitoring for various AWS services and common workloads, such as Java Virtual Machines (JVM), Apache Kafka, Apache Tomcat, and NGINX. These solutions provide focused guidance on key monitoring tasks, including the installation and configuration of the CloudWatch agent, deployment of pre-defined custom dashboards, and setup of metric alarms. They are designed to assist developers and operations teams in leveraging AWS monitoring and observability tools more effectively.

The solutions include guidance on when to use specific observability features like Detailed Monitoring metrics for infrastructure, Container Insights for container monitoring, and Application Signals for application monitoring. By providing working examples and practical configurations, these solutions aim to simplify the initial setup process, allowing you to establish functional monitoring more quickly and customize as needed for their specific requirements.

To get started with observability solutions, visit the observability solutions page in the CloudWatch console.

For open-source solutions that work with Amazon Managed Grafana, see Amazon Managed Grafana solutions

The available solutions are detailed below. Some solutions use the CloudWatch agent, and others use a Amazon CloudWatch managed Prometheus collector to scrape Prometheus-compatible metrics and deliver them to CloudWatch.

Topics

- CloudWatch solution: JVM workload on Amazon EC2
- CloudWatch solution: NGINX workload on Amazon EC2
- CloudWatch solution: NVIDIA GPU workload on Amazon EC2
- CloudWatch solution: Kafka workload on Amazon EC2
- CloudWatch solution: Tomcat workload on Amazon EC2
- CloudWatch solution: Amazon EC2 health
- Amazon CloudWatch solution: Prometheus metric collection on Amazon Amazon EC2
- Amazon CloudWatch solution: Prometheus metric collection on Amazon ECS

- Amazon CloudWatch solution: Amazon MSK insights
- Amazon CloudWatch solution: Amazon OpenSearch Service insights

How do solution dashboards work?

The dashboards for CloudWatch solutions use search-powered variables (dropdowns) that allow you to explore and visualize different aspects of your workloads dynamically.

By combining the flexibility of search-powered variables with the pre-configured metric widgets, the dashboard provides deep insights into your workloads, enabling proactive monitoring, troubleshooting, and optimization. This dynamic approach ensures that you can quickly adapt the dashboard to your specific monitoring needs, without the need for extensive customization or configuration.

Do solutions support cross-Region observability?

CloudWatch solution dashboards display metrics of the Region where the solution dashboard is created. However, the solution dashboard doesn't display metrics across multiple Regions. If you have a use case to view data from multiple Regions in a single dashboard, you'll need to customize the dashboard JSON to add the Regions that you want to view. To do this, use the `region` attribute of the metric format to query the metrics from different Regions. For more information about modifying dashboard JSON, see Metric Widget: Format for Each Metric in the Array.

Do solution dashboards support Cross-account cross-Region CloudWatch console?

When using CloudWatch cross-account observability, solution dashboards in the central monitoring account display metrics from source accounts in the same Region. To differentiate metrics for similar workloads across accounts, provide unique grouping dimension values in agent configurations. For instance, assign distinct `ClusterName` values for Kafka brokers in different accounts for Kafka workload, enabling precise cluster selection and metric viewing in the dashboard.

Do solution dashboards support CloudWatch cross-account observability?

If you have enabled cross-account using Cross-account cross-Region CloudWatch console, you won't be able to use the solution dashboard created in the monitoring account to view metrics from source accounts. Instead, you'll need to create dashboards in the respective source accounts. However, you can create the dashboard in the source account and view it from the monitoring account by switching the account ID setting in the console.

What are the limitations for a solution dashboard?

Solution dashboards leverage Search expressions to filter and analyze metrics for the workloads. This enables dynamic views based on dropdown option selections. These search expressions might return more than 500 time series, but each dashboard widget can't display more than 500 time series. If a metric search in the solution dashboard results in more than 500 time series across all Amazon EC2; instances, the graph displaying the top contributors might show inaccurate results. For more information about search expressions, see CloudWatch search expression syntax.

CloudWatch displays the metric information on the dashboards if you click the `i` icon on the dashboard widget. However, this currently doesn't work for dashboard widgets that use search expressions. The solution dashboards use search expressions, so you won't be able to see the metric description in the dashboard.

Can I customize the agent configuration or the dashboard provided by a solution?

You can customize the agent configuration and the dashboard. Be aware that if you customize the agent configuration, you must update the dashboard accordingly or it will display empty metric widgets. Also be aware that if CloudWatch releases a new version of a solution, you might have to repeat your customizations if you apply the newer version of the solution.

How are solutions versioned?

Each solution provides the most up-to-date instructions and resources. Always use the latest version available. While the solutions themselves are not versioned, the associated artifacts (such as CloudFormation templates for dashboards and agent installations) are versioned.

You can identify the version of a previously deployed artifact by checking the CloudFormation template's description field or the filename of the template you downloaded. To determine if you're using the latest version, compare your deployed version with the one currently referenced in the solution documentation.

CloudWatch solution: JVM workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for JVM application running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements
- Benefits
- Costs
- CloudWatch agent configuration for this solution
- Deploy the agent for your solution
- Create the JVM solution dashboard

Requirements

This solution is relevant for the following conditions:

- Supported versions: Java LTS versions 8, 11, 17, and 21
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all JVM workloads in a given AWS Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers JVM monitoring, providing valuable insights for the following use cases:

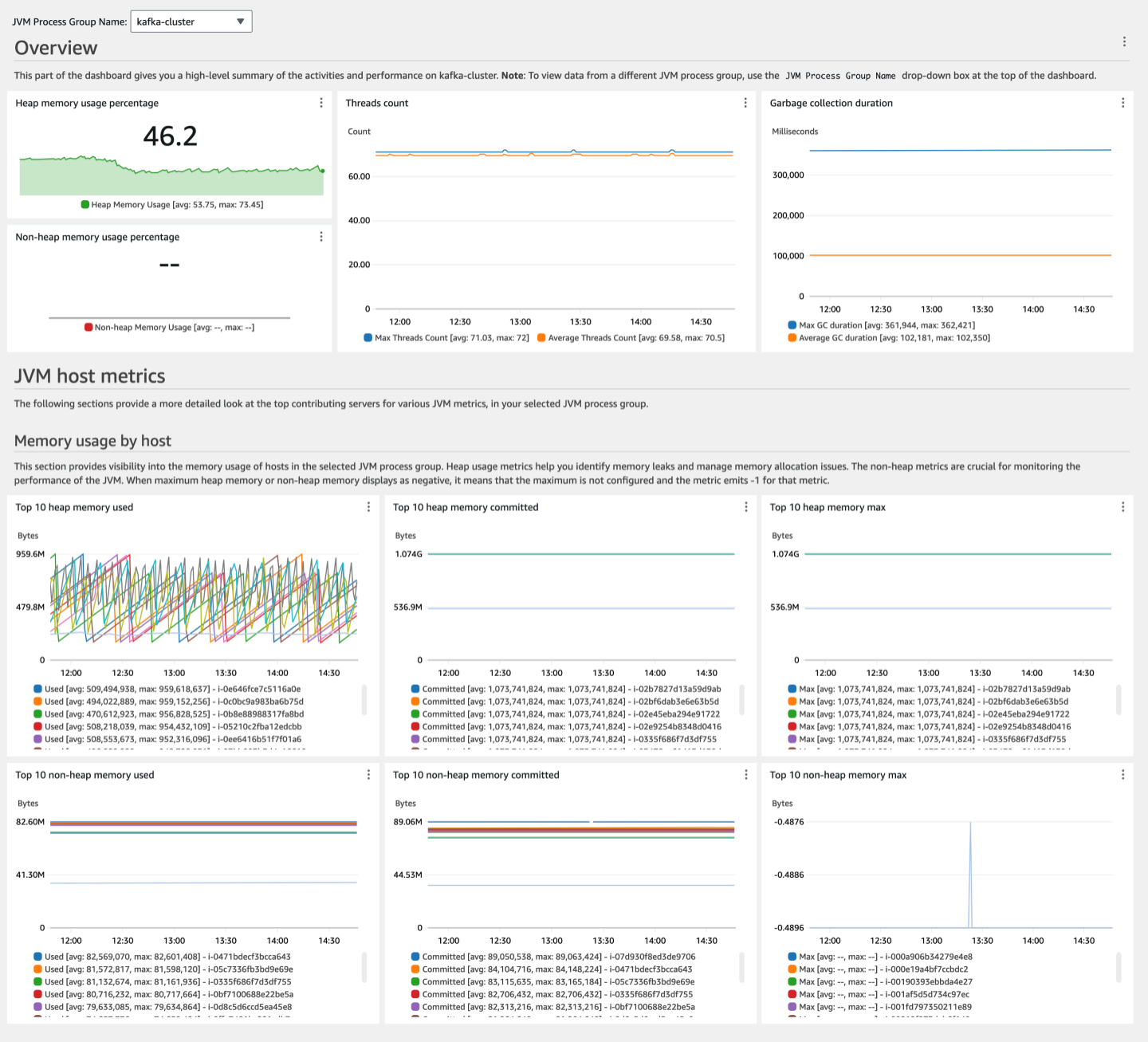
- Monitor JVM heap and non-heap memory usage.
- Analyze thread and class loading for concurrency issues.

- Track garbage collection to identify memory leaks.
- Switch between different JVM applications configured through the solution under the same account.

Below are the key advantages of the solution:

- Automates metric collection for JVM using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for JVM metrics. The dashboard will automatically handle metrics from new JVM EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.



Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.

- Each JVM host configured for the solution publishes a total of 18 metrics plus one metric (`disk_used_percent`) for which the metric count depends on the number of paths for the host.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. For **Choose a Region**, select the Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`(18 + average number of disk paths per EC2 host) * number of EC2 instances configured for this solution`**.
4. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.
5. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
6. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The agent configuration in this solution collects the foundational metrics for the solution. The CloudWatch agent can be configured to collect more JVM metrics than the dashboard displays by default. For a list of all JVM metrics that you can collect, see Collect JVM metrics. For general information about CloudWatch agent configuration, see Metrics collected by the CloudWatch agent.

Expose JMX ports for the JVM application

The CloudWatch agent relies on JMX to collect the metrics related to the JVM process. To make this possible, you must expose the JMX port from your JVM application. Instructions for exposing the JMX port depend on the workload type you are using for your JVM application. See the documentation for your application to find these instructions.

In general, to enable a JMX port for monitoring and management, you would set the following system properties for your JVM application. Be sure to specify an unused port number. The following example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the JMX documentation to set the required property.

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=port-number
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

Review the starting scripts and configuration files of your application to find the best place to add these arguments. When you run a `.jar` file from the command line, this command could look like the following, where *`pet-search.jar`* is the name of the application jar.

```
$ java -jar -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=9999 -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false pet-search.jar
```

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. It is in the `endpoint` line in the configuration.
- `ProcessGroupName`– Provide meaningful names for the `ProcessGroupName` dimension. These names should represent the cluster, application, or services grouping for EC2 instances running the same application or process. This helps you to group metrics from instances belonging to the same JVM process group, providing a unified view of cluster, application, and service performance in the solution dashboard.

For example, if you have two Java applications running in the same account, one for the `order-processing` application and another for the `inventory-management` application, you should set the `ProcessGroupName` dimensions accordingly in the agent configuration of each instance.

- For the `order-processing` application instances, set `ProcessGroupName=order-processing`.
- For the `inventory-management` application instances, set `ProcessGroupName=inventory-management`.

When you follow these guidelines, the solution dashboard will automatically group the metrics based on the `ProcessGroupName` dimension. The dashboard will include dropdown options to select and view metrics for a specific process group, allowing you to monitor the performance of individual process groups separately.

Agent configuration for JVM hosts

Use the following CloudWatch agent configuration on EC2 instances where your Java applications is deployed. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

Replace *`ProcessGroupName`* with the name of your process group. Replace *`port-number`* with the JMX port of your Java application. If JMX was enabled with password authentication or SSL

for remote access, see Collect Java Management Extensions (JMX) metrics for information about setting up TLS or authorization in agent configuration as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "ProcessGroupName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
```

```
        "measurement": [
          "used_percent"
        ]
      },
      "swap": {
        "measurement": [
          "used_percent"
        ]
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_time_wait"
        ]
      }
    }
  }
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of JVM with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances.

You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.
   a. Choose **Create parameter**.
   b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-JVM-Configuration`**.
   c. (Optional) In the **Description** box, type a description for the parameter.
   d. For **Parameter tier**, choose **Standard**.
   e. For **Type**, choose **String**.
   f. For **Data type**, choose **text**.
   g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for JVM hosts. Be sure to customize the grouping dimension value and port number as described.
   h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use AWS CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.

4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-JVM-Configuration`**.

   b. To select the target instances, you have two options.

      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

      If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .

Note

After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:

1. If the Systems Manager parameter is deleted, the agent will stop.
2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.

3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Make sure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

If everything is set up correctly, then you should see the JVM metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that JVM metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in Agent configuration for JVM hosts, such as `jvm.memory.heap.used`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the JVM solution dashboard

The dashboard provided by this solution presents metrics for the underlying Java Virtual Machine (JVM) for the server. It offers an overview of the JVM by aggregating and presenting metrics across all instances, providing a high-level summary of the overall health and operational state. Additionally, the dashboard shows a breakdown of the top contributors (top 10 per metric widget)

for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for AWS services, see Viewing a CloudWatch dashboard for a single AWS service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your JVM metrics are published.
> If CloudWatch agent metrics are getting published to a different namespace than `CWAgent` (for example, if you've provided a customized namespace), you'll have to change the CloudFormation configuration to replace `CWAgent` with the customized namespace you are using.

To create the dashboard through CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage

collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=JvmOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`JVMDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/JVM_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the JVM workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`JVMDashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`JVMDashboard-us-east-1`**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.

7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/JVM_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the JVM dashboard

Here are a few tasks that you can try out with the new JVM dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a JVM process group. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Select a process group

Use the **JVM Process Group Name** dropdown list to select the process group that you want to monitor. The dashboard automatically updates to display metrics for the selected process group. If you have multiple Java applications or environments, each might be represented as a separate process group. Selecting the appropriate process group ensures that you're viewing metrics specific to the application or environment that you want to analyze.

Review memory usage

From the dashboard overview section, find the **Heap Memory Usage Percentage** and **Non-Heap Memory Usage Percentage** widgets. These show the percentage of heap and non-heap memory being used across all JVMs in the selected process group. A high percentage indicates potential memory pressure that could lead to performance issues or `OutOfMemoryError` exceptions. You can also drill down to heap usage by host under **Memory usage by host** to check the hosts with high usage.

Analyze threads and classes loaded

In the **Threads and Classes Loaded by Host** section, find the **Top 10 Threads Count** and **Top 10 Classes Loaded** widgets. Look for any JVMs with an abnormally high number of threads or classes compared to others. Too many threads can indicate thread leaks or excessive concurrency, while a large number of loaded classes could point to potential class loader leaks or inefficient dynamic class generation.

Identify garbage collection issues

In the **Garbage Collection** section, find the **Top 10 Garbage Collections Invocations Per Minute** and **Top 10 Garbage Collection Duration** widgets for the different garbage collector types: **Young**, **Concurrent**, and **Mixed**. Look for any JVMs that have an unusually high number of collections or long collection durations compared to others. This could indicate configuration issues or memory leaks.

CloudWatch solution: NGINX workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for NGINX application running on EC2 instances. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements
- Benefits
- Costs
- CloudWatch agent configuration for this solution
- Deploy the agent for your solution
- Create the NGINX solution dashboard

Requirements

This solution is relevant for the following conditions:

- Supported versions: NGINX version 1.24
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all NGINX workloads in a given AWS Region
- Latest version of CloudWatch agent
- Prometheus Exporter: nginxinc/nginx-prometheus-exporter (Apache 2.0 license)
- SSM agent installed on EC2 instance

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

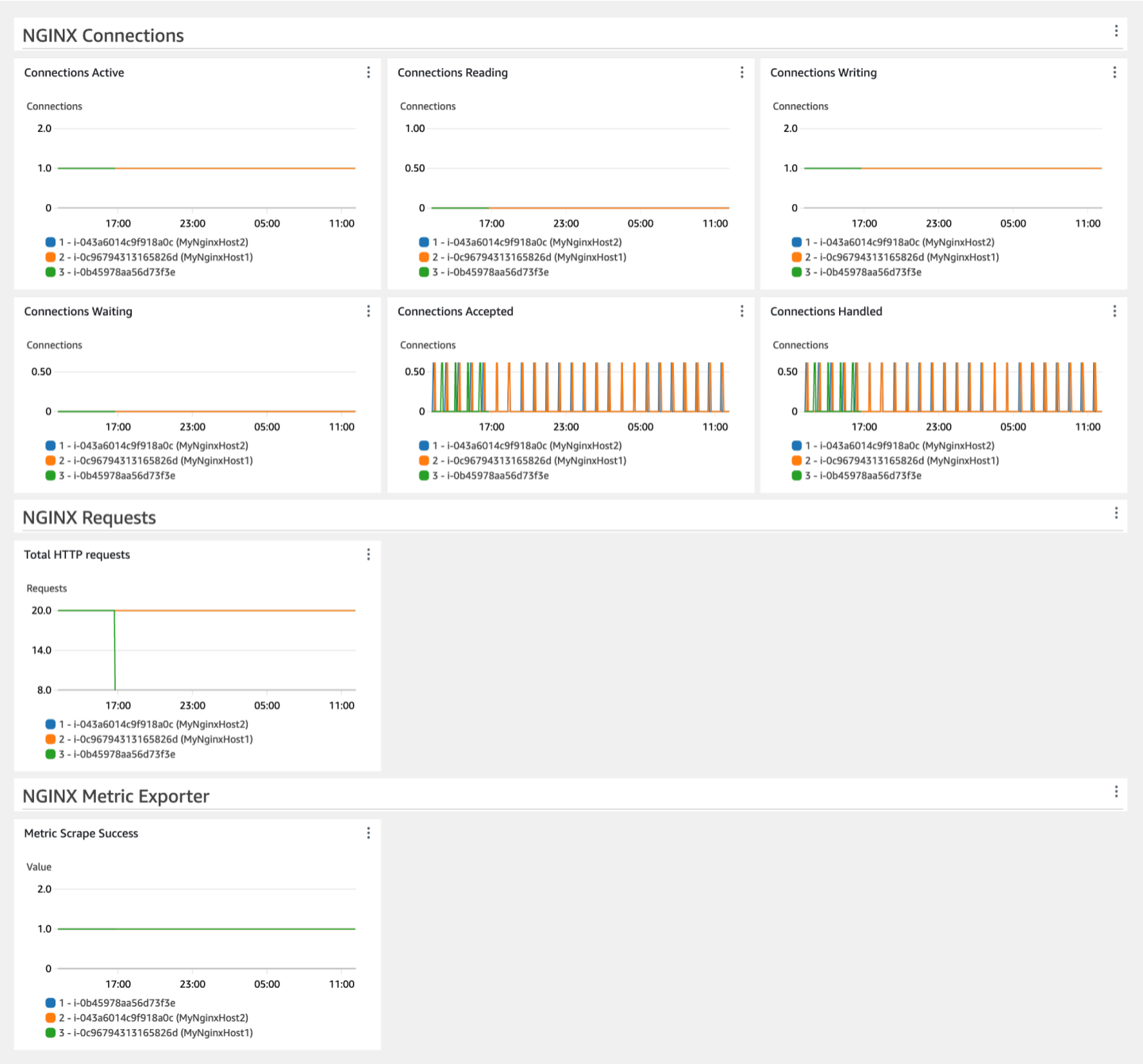
The solution delivers NGINX monitoring, providing valuable insights for the following use cases:

- Review connection metrics to identify potential bottlenecks, connection issues, or unexpected usage.
- Analyze HTTP request volume to understand overall traffic load on the NGINX.

Below are the key advantages of the solution:

- Automates metric collection for NGINX using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for NGINX metrics. The dashboard will automatically handle metrics from new NGINX EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard.

The following image is an example of the dashboard for this solution.



Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent for this solution are published to CloudWatch Logs using the Embedded Metric Format (EMF). These CloudWatch logs are charged based on

their volume and retention period. Therefore, you will not be billed for any **PutMetricData** API calls for this solution. The metrics extracted and ingested from your logs are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.

  - Each NGINX EC2 host configured for the solution publishes a total of eight metrics.
- One custom dashboard.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. For **Choose a Region**, select the AWS Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`8 * number of EC2 instances configured for this solution`**.
4. In the **Logs** section, for **Standard Logs: Data Ingested**, enter the estimated daily log volume generated by the CloudWatch Agent across all EC2 hosts. For example, five EC2 instances produce less than 1000 bytes per day. Once set up, you can check your byte usage using the `IncomingBytes` metric, vended by CloudWatch Logs. Be sure to select the appropriate log group.
5. In the **Logs** section, for **Log Storage/Archival (Standard and Vended Logs)**, select **`Yes to Store Logs: Assuming 1 month retention`**. Modify this value if you decide to make custom changes to the retention period.
6. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
7. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your NGINX workload. The CloudWatch agent can be configured to collect more NGINX metrics than the dashboard displays by default. For a list of all NGINX metrics that you can collect, see Metrics for NGINX OSS.

Before configuring the CloudWatch agent, you must first configure NGINX to expose its metrics. Secondly, you must install and configure the third party Prometheus metric exporter.

Expose NGINX metrics

> **Note**
>
> The following commands are for Linux. Check NGINX for Windows page for equivalent commands in Windows Server.

You must first enable the `stub_status` module. Add a new location block in your NGINX configuration file. Add the following lines in the `server` block of your `nginx.conf` to enable NGINX's `stub_status` module:

```
location /nginx_status {
          stub_status on;
          allow 127.0.0.1; # Allow only localhost to access
          deny all; # Deny all other IPs
        }
```

Before reloading NGINX, validate your NGINX configuration:

```
sudo nginx -t
```

This validation command helps to prevent any unforeseen errors, which can cause your website to gown down. The following example demonstrates a successful response:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

After you've successfully validated the updated configuration, reload NGINX (no output is expected):

```
sudo systemctl reload nginx
```

This command instructs the NGINX process to reload the configuration. Reloads are more graceful compared to a full restart. A reload starts the new worker process with a new configuration, gracefully shutting down old worker processes.

Test the NGINX status endpoint:

```
curl http://127.0.0.1/nginx_status
```

The following example demonstrates a successful response:

```
Active connections: 1
server accepts handled requests
6 6 6
Reading: 0 Writing: 1 Waiting: 0
```

The following example demonstrates a failure response (review the previous steps before proceeding):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/
xhtml11.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
      <head>
          <title>The page is not found</title>
          ...
```

Configure Prometheus metric exporter

Download the latest NGINX Prometheus exporter release from the official GitHub repository. You must download the relevant binary for your platform.

The following example demonstrates commands for AMD64:

```
cd /tmp
wget https://github.com/nginxinc/nginx-prometheus-exporter/releases/download/v1.3.0/
nginx-prometheus-exporter_1.3.0_linux_amd64.tar.gz
tar -xzvf nginx-prometheus-exporter_1.3.0_linux_amd64.tar.gz
sudo cp nginx-prometheus-exporter /usr/local/bin/
rm /tmp/nginx-prometheus-exporter*
```

Run the Prometheus exporter and point it to the NGINX stub status page:

```
nohup /usr/local/bin/nginx-prometheus-exporter -nginx.scrape-uri http://127.0.0.1/
nginx_status &>/dev/null &
```

The following example demonstrates a response (background job ID and PID):

```
[1] 74699
```

Test the NGINX Prometheus endpoint

Validate that the NGINX Prometheus exporter has started to expose the relevant metrics:

```
curl http://localhost:port-number/metrics
```

The following example demonstrates a successful response:

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection
 cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
...
# HELP nginx_connections_accepted Accepted client connections
# TYPE nginx_connections_accepted counter
nginx_connections_accepted 14
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 1
...
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 1
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the agent and Prometheus configurations for your environment such as the port number used by the Prometheus exporter.

The port used by the Prometheus exporter can be verified using the following command:

```
sudo netstat -antp | grep nginx-prom
```

The following example demonstrates a response (see port value 9113):

```
tcp6 0 0 :::9113 :::* LISTEN 76398/nginx-prometh
```

Agent configuration for NGINX hosts

The CloudWatch agent with Prometheus monitoring needs two configurations to scrape the Prometheus metrics. Each configuration will be stored as a separate parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

The first configuration is for the Prometheus exporter, as documented in Prometheus' scrape_config documentation. The second configuration is for the CloudWatch agent.

Prometheus configuration

Replace *port-number* with the port of your server.

```
global:
  scrape_interval: 30s
  scrape_timeout: 10s

scrape_configs:
- job_name: 'nginx'
  metrics_path: /metrics
  static_configs:
    - targets: ['localhost:port-number']
  ec2_sd_configs:
    - port: port-number
  relabel_configs:
    - source_labels: ['__meta_ec2_instance_id']
```

```
      target_label: InstanceId
  metric_relabel_configs:
    - source_labels: ['__name__']
      regex: 'nginx_up|nginx_http_requests_total|nginx_connections_.*'
      action: keep
```

CloudWatch agent configuration

As per the previous CloudWatch agent configuration, these metrics are published through CloudWatch Logs using the embedded metric format (EMF). These logs are configured to use the log group `nginx`. You can customize the *`log_group_name`* with a different name that represents the CloudWatch logs.

If you are using Windows Server, set *`prometheus_config_path`* in the following configuration to `C:\\ProgramData\\Amazon\\AmazonCloudWatchAgent\\prometheus.yaml`.

```
{
"agent": {
  "metrics_collection_interval": 60
},
"logs": {
  "metrics_collected": {
      "prometheus": {
          "log_group_name": "nginx",
          "prometheus_config_path": "/opt/aws/amazon-cloudwatch-agent/etc/
prometheus.yaml",
          "emf_processor": {
              "metric_declaration_dedup": true,
              "metric_namespace": "CWAgent",
              "metric_declaration":[
                  {
                      "source_labels":["InstanceId"],
                      "metric_selectors":["nginx_up", "nginx_http_requests_total",
 "nginx_connections*"],
                      "dimensions": [["InstanceId"]]
                  }
              ]
          }
      }
  }
}
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. You recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge new CloudWatch agent and Prometheus configurations with your existing configurations, and then deploy the merged configurations. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. You must also grant the CloudWatch agent EC2 read access. EC2 read access is required for the EC2 InstanceId to be added as a metric dimension. This additional requirement is driven by `prometheus.yaml` as detailed above because it uses `__meta_ec2_instance_id` through EC2 Service Discovery.

Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy**, **AmazonSSMManagedInstanceCore**, and **AmazonEC2ReadOnlyAccess** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. Verify that the selected Region on the console is the Region where NGINX is running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**
4. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

    b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-NGINX-CloudWatchAgent-Configuration`**.

c. (Optional) In the **Description** box, type a description for the parameter.

d. For **Parameter tier**, choose **Standard**.

e. For **Type**, choose **String**.

f. For **Data type**, choose **text**.

g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for NGINX hosts. Be sure to customize as required. For example, the relevant `log_group_name`.

h. Choose **Create parameter**.

To create the Prometheus configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**
3. Follow these steps to create a new parameter for the configuration.

   a. Choose **Create parameter**.

   b. In the **Name** box, enter a name that you'll use to reference the configuration file in later steps. For example, **`AmazonCloudWatch-NGINX-Prometheus-Configuration`**.

   c. (Optional) In the **Description** box, type a description for the parameter.

   d. For **Parameter tier**, choose **Standard**.

   e. For **Type**, choose **String**.

   f. For **Data type**, choose **text**.

   g. In the **Value** box, paste the corresponding YAML block that was listed in the Agent configuration for NGINX hosts. Be sure to customize as required. For example, the relevant port number as per `targets`.

   h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use AWS CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-with-prometheus-config-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:
   a. For **CloudWatchAgentConfigSSM**, enter the name of the AWS Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NGINX-CloudWatchAgent-Configuration`**.
   b. For **PrometheusConfigSSM**, enter the name of the AWS Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NGINX-Prometheus-Configuration`**.
   c. To select the target instances, you have two options.
      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.
      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)
5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-with-prometheus-config-1.0.0.json.

Note

After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:

1. If the Systems Manager parameter is deleted, the agent will stop.
2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Make sure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

If everything is set up correctly, then you should see the NGINX metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that NGINX metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for metrics such as `nginx_http_requests_total`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the NGINX solution dashboard

The dashboard provided by this solution presents NGINX workload metrics by aggregating and presenting metrics across all instances. The dashboard shows a breakdown of the top contributors (top 10 per metric widget) for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your NGINX metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard through CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=NginxOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NGINXDashboard-us-east-1`**.

4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NGINX_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the NGINX workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`NGINXDashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.
5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NGINXDashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NGINX_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

Get started with the NGINX dashboard

Here are a few tasks that you can try out with the new NGINX dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a NGINX workload. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Review connection metrics

In the **Connections** section, you can find several key metrics that provide insights into the client connection handling of your NGINX server. Monitoring these connection metrics can help you identify potential bottlenecks, connection issues, or unexpected connection patterns.

- Accepted client connections
- Active client connections
- Handled client connections
- Connections reading requests
- Idle client connections
- Connections writing responses

Analyze HTTP request volume

The `request` metric in the **HTTP Requests** section shows the total number of HTTP requests handled by the NGINX server. Tracking this metric over time can help you understand the overall traffic load on your NGINX infrastructure and plan for resource allocation and scaling accordingly.

CloudWatch solution: NVIDIA GPU workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for NVIDIA GPU workloads running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements
- Benefits
- CloudWatch agent configuration for this solution

- Deploy the agent for your solution
- Create the NVIDIA GPU solution dashboard

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon EC2
- Supports up to 500 GPUs across all EC2 instances in a given AWS Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance
- The EC2 instance must have an NVIDIA driver installed. NVIDIA drivers are pre-installed on some Amazon Machine Images (AMIs). Otherwise, you can manually install the driver. For more information, see Install NVIDIA drivers on Linux instances.

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers NVIDIA monitoring, providing valuable insights for the following use cases:

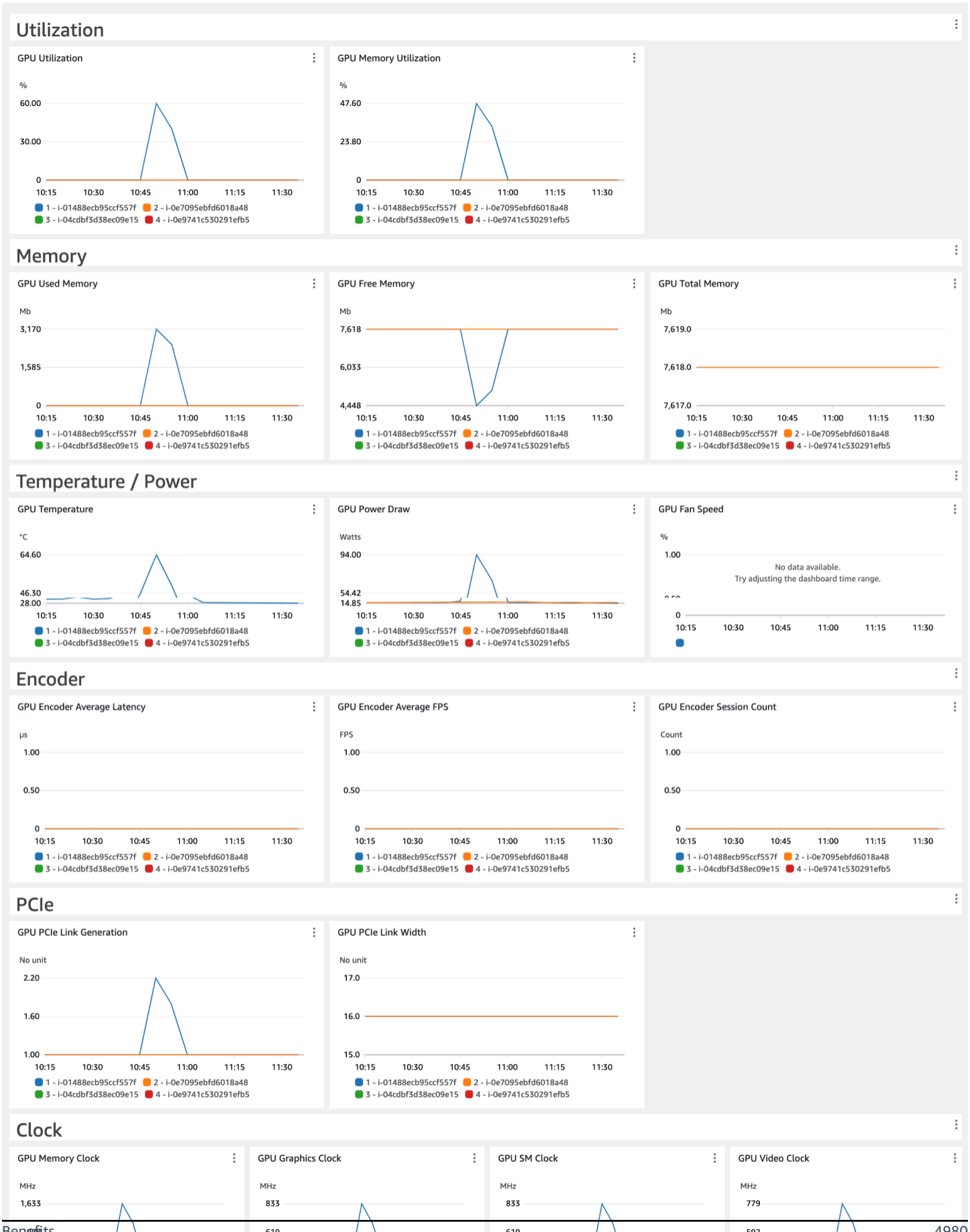
- Analyze GPU and memory usage for performance bottlenecks or the need for additional resources.
- Monitor temperature and power draw to make sure GPUs operate within safe limits.
- Evaluate encoder performance for GPU video workloads.
- Verify PCIe connectivity for expected generation and width.

- Monitor GPU clock speeds to detect scaling and throttling issues.

Below are the key advantages of the solution:

- Automates metric collection for NVIDIA using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for NVIDIA metrics. The dashboard will automatically handle metrics from new NVIDIA EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard.

The following image is an example of the dashboard for this solution.


Utilization
GPU Utilization
%
60.00
30.00
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Memory Utilization
%
47.60
23.80
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
Memory
GPU Used Memory
Mb
3,170
1,585
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Free Memory
Mb
7,618
6,033
4,448
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Total Memory
Mb
7,619.0
7,618.0
7,617.0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
Temperature / Power
GPU Temperature
°C
64.60
46.30
28.00
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Power Draw
Watts
94.00
54.42
14.85
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Fan Speed
%
1.00
No data available.
Try adjusting the dashboard time range.
0
10:15 10:30 10:45 11:00 11:15 11:30
Encoder
GPU Encoder Average Latency
µs
1.00
0.50
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Encoder Average FPS
FPS
1.00
0.50
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU Encoder Session Count
Count
1.00
0.50
0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
PCIe
GPU PCIe Link Generation
No unit
2.20
1.60
1.00
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
GPU PCIe Link Width
No unit
17.0
16.0
15.0
10:15 10:30 10:45 11:00 11:15 11:30
1 - i-01488ecb95ccf557f 2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15 4 - i-0e9741c530291efb5
Clock
GPU Memory Clock
MHz
1,633
324
10:30 11:00 11:30
1 - i-01488ecb95ccf557f
2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15
4 - i-0e9741c530291efb5
GPU Graphics Clock
MHz
833
619
405
10:30 11:00 11:30
1 - i-01488ecb95ccf557f
2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15
4 - i-0e9741c530291efb5
GPU SM Clock
MHz
833
619
405
10:30 11:00 11:30
1 - i-01488ecb95ccf557f
2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15
4 - i-0e9741c530291efb5
GPU Video Clock
MHz
779
592
405
10:30 11:00 11:30
1 - i-01488ecb95ccf557f
2 - i-0e7095ebfd6018a48
3 - i-04cdbf3d38ec09e15
4 - i-0e9741c530291efb5

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each EC2 host configured for the solution publishes a total of 17 metrics per GPU.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. For **Choose a Region**, select the Region where you would like to deploy the solution.
3. In the **Metrics** section, for **Number of metrics**, enter **`17 * average number of GPUs per EC2 host * number of EC2 instances configured for this solution`**.
4. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.
5. By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.
6. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
7. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your NVIDIA GPU. The CloudWatch agent can be configured to collect more NVIDIA GPU metrics than the dashboard displays by default. For a list of all NVIDIA GPU metrics that you can collect, see Collect NVIDIA GPU metrics.

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

Use the following CloudWatch agent configuration on EC2 instances with NVIDIA GPUs. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

```
{
    "metrics": {
        "namespace": "CWAgent",
        "append_dimensions": {
            "InstanceId": "${aws:InstanceId}"
        },
        "metrics_collected": {
            "nvidia_gpu": {
                "measurement": [
                    "utilization_gpu",
                    "temperature_gpu",
                    "power_draw",
                    "utilization_memory",
                    "fan_speed",
                    "memory_total",
                    "memory_used",
                    "memory_free",
                    "pcie_link_gen_current",
                    "pcie_link_width_current",
                    "encoder_stats_session_count",
                    "encoder_stats_average_fps",
                    "encoder_stats_average_latency",
                    "clocks_current_graphics",
                    "clocks_current_sm",
```

```
                    "clocks_current_memory",
                    "clocks_current_video"
                ],
                "metrics_collection_interval": 60
            }
        }
    },
    "force_flush_interval": 60
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of NVIDIA GPU with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.

- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**.
4. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-NVIDIA-GPU-Configuration`**.

c. (Optional) In the **Description** box, type a description for the parameter.

d. For **Parameter tier**, choose **Standard**.

e. For **Type**, choose **String**.

f. For **Data type**, choose **text**.

g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for this solution.

h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use AWS CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-NVIDIA-GPU-Configuration`**.

   b. To select the target instances, you have two options.

i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console.

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
> 3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Make sure the target EC2 instances have the required IAM permissions.

- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

If everything is set up correctly, then you should see the NVIDIA GPU metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that NVIDIA GPU metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in Agent configuration for this solution, such as `nvidia_smi_utilization_gpu`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the NVIDIA GPU solution dashboard

The dashboard provided by this solution presents NVIDIA GPUs metrics by aggregating and presenting metrics across all instances. The dashboard shows a breakdown of the top contributors (top 10 per metric widget) for each metric. This helps you to quickly identify outliers or instances that significantly contribute to the observed metrics.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your NVIDIA GPU metrics are published.
> If you've specified a custom namespace other than CWAgent in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace CWAgent with the customized namespace you are using.

To create the dashboard through CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=NvidiaGpuOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NVIDIA-GPU-Dashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NVIDIA_GPU_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the NVIDIA GPU workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`NVIDIA-GPU-DashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`NVIDIA-GPU-Dashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/NVIDIA_GPU_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json .

Get started with the NVIDIA GPU dashboard

Here are a few tasks that you can try out with the new NVIDIA GPU dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor your NVIDIA GPUs. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Review GPU utilization

From the **Utilization** section, find the **GPU Utilization** and **Memory Utilization** widgets. These show the percentage of time the GPU is being actively used for computations and the percentage of global memory being read or written, respectively. High utilization could indicate potential performance bottlenecks or the need for additional GPU resources.

Analyze GPU memory usage

In the **Memory** section, find the **Total Memory**, **Used Memory**, and **Free Memory** widgets. These provide insights into the overall memory capacity of the GPUs and how much memory is currently being consumed or available. Memory pressure could lead to performance issues or out-of-memory errors, so it's important to monitor these metrics and make sure sufficient memory is available for your workloads.

Monitor temperature and power draw

In the **Temperature / Power** section, find the **GPU Temperature** and **Power Draw** widgets. These metrics are essential for ensuring that your GPUs are operating within safe thermal and power limits.

Identify encoder performance

In the **Encoder** section, find the **Encoder Session Count**, **Average FPS**, and **Average Latency** widgets. These metrics are relevant if you're running video encoding workloads on your GPUs. Monitor these metrics to make sure that your encoders are performing optimally and identify any potential bottlenecks or performance issues.

Check PCIe link status

In the **PCIe** section, find the **PCIe Link Generation** and **PCIe Link Width** widgets. These metrics provide information about the PCIe link connecting the GPU to the host system. Make sure that the link is operating at the expected generation and width to avoid potential performance limitations due to PCIe bottlenecks.

Review GPU clocks

In the **Clock** section, find the **Graphics Clock**, **SM Clock**, **Memory Clock**, and **Video Clock** widgets. These metrics show the current operating frequencies of various GPU components. Monitoring these clocks can help identify potential issues with GPU clock scaling or frequency throttling, which could impact performance.

CloudWatch solution: Kafka workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for Kafka workloads (brokers, producers, and consumers) running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements
- Benefits
- Costs
- CloudWatch agent configuration for this solution

- Deploy the agent for your solution
- Create the Kafka solution dashboard
- Configure the agent for multiple Kafka roles on the same instance

Requirements

This solution is relevant for the following conditions:

- Workload: Kafka v0.8.2.x and later
- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all Kafka workloads in a given AWS Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

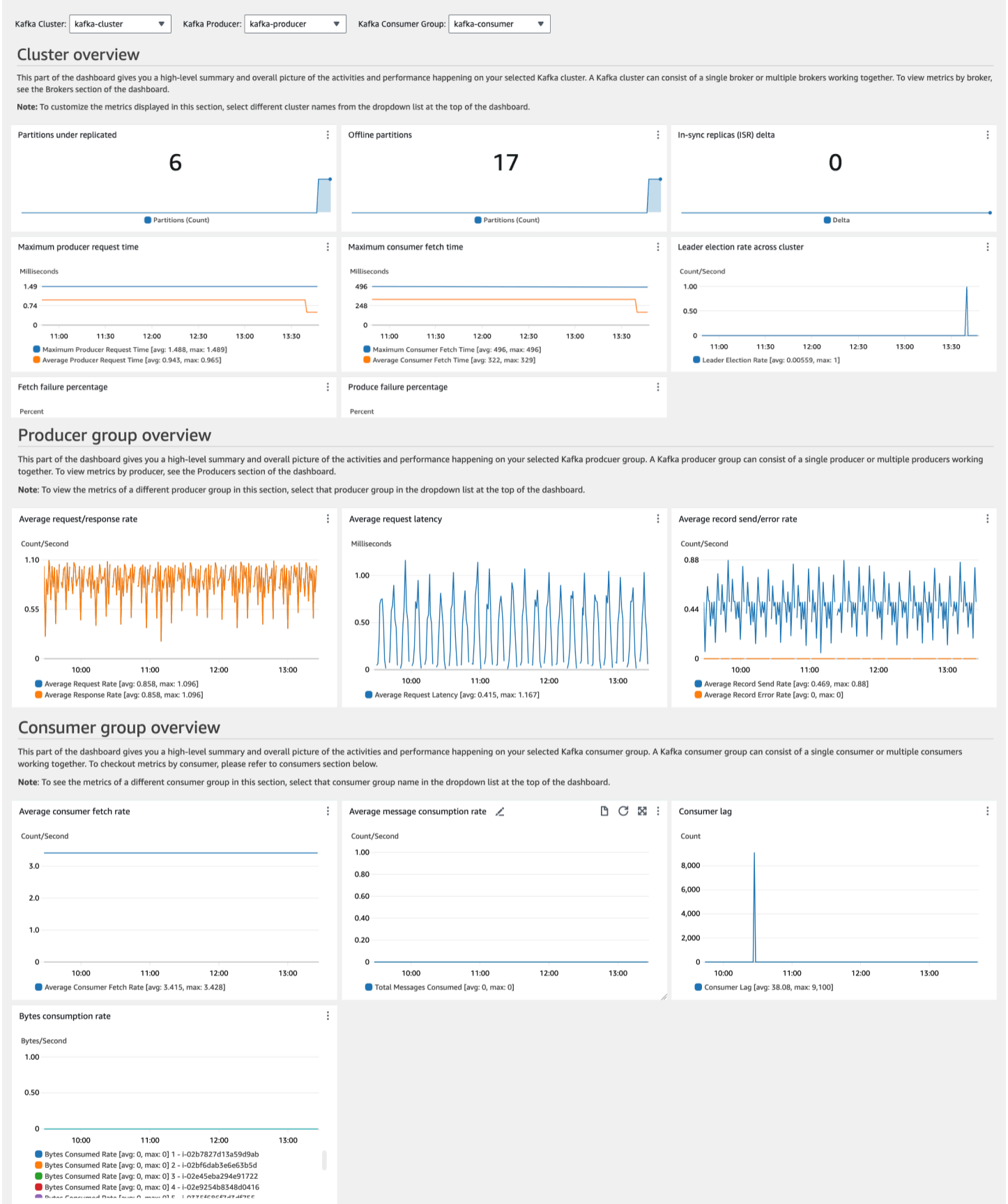
The solution delivers Kafka server monitoring, providing valuable insights for the following use cases:

- Monitor Kafka cluster health through replication and sync metrics.
- Track broker performance through request failures and latencies along with network traffic.
- Monitor producer/consumer errors, latencies, and consumer lag.
- Analyze underlying JVM performance for Kafka clusters.
- Switch between multiple Kafka clusters, producers, and consumers configured through the solution under the same account.

Below are the key advantages of the solution:

- Automates metric collection for Kafka and the underlying JVM using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for Kafka and JVM metrics. The dashboard will automatically handle metrics from new Kafka EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.



End of preview

Create dashboard to view in full

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each broker host configured for the solution publishes 33 metrics plus one metric (`disk_used_percent`) for which the metric count for each EC2 host depends on the number of disk paths for that host.
  - Each producer host configured for the solution publishes three metrics with the `topic` dimension and three metrics without the `topic` dimension. For the metrics with the `topic` dimension, each topic counts as a separate metric.
  - Each consumer host configured for the solution publishes two metrics with `topic` dimensions and three metrics without `topic` dimensions. For the metrics with topic dimensions, each topic counts as a separate metric.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute for each EC2 host. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. In the **Metrics** section, for **Number of metrics**, enter **`broker_metrics_count + producer_metrics_count + consumer_metrics_count`**. Calculate these as follows:
   - `broker_metrics_count` = (33 + average number of disk paths per EC2 host) * number_of_ec2_broker_hosts
   - `producer_metrics_count` = (3 * average_number_of_topics_per_producer_host + 3) * number_of_ec2_producer_hosts

- `consumer_metrics_count` = (2 * average_number_of_topics_per_consumer_host + 3) * number_of_ec2_consumer_hosts

3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the CloudWatch agent performs one **PutMetricData** operation each minute for each EC2 host.

4. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
5. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The agent configuration in this solution collects the foundational metrics for Kafka, JVM, and EC2. The CloudWatch agent can be configured to collect more Kafka and JVM metrics than the dashboard displays by default. For a list of all Kafka metrics that you can collect, see Collect Kafka metrics. For a list of all JVM metrics that you can collect, see Collect JVM metrics. For a list of EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

Expose JMX ports for the Kafka broker, producer, and consumer roles

The CloudWatch agent relies on JMX to collect the metrics related to the Kafka brokers, producers, and consumers. To make this possible, you must expose the JMX port on your servers and applications.

For Kafka brokers, you must use the `JMX_PORT` environment variable to set the port. You'll have to restart the brokers after you set this environment variable. Review the starting scripts and configuration files of your application to find the best place to add these arguments.

For example, for Linux and macOS systems, you can use the following command to set the JMX port. Be sure to specify an unused port number.

```
export JMX_PORT=port-number
```

For Kafka producers and consumers, instructions for exposing the JMX port depend on the workload type you are using for your producer or consumer JVM application. See the documentation for your application to find these instructions.

In general, to enable a JMX port for monitoring and management, you would set the following system properties for your JVM application. The following example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the JMX documentation to set the required property.

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=port-number
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

To verify the JMX port, run `ps aux | grep jmxremote.port`. The results should show that the JMX port was set on the JVM processes.

Agent configuration for this solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard. Each Kafka role, such as broker, producer, or consumer, has its own agent configuration that enables the collection of Kafka metrics and underlying JVM and EC2 metrics.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. The port number is in the `endpoint` line in the configuration.
- `ClusterName`– This is used as a dimension for broker metrics collected. Provide a meaningful name that represents the cluster grouping for the instances that run the Kafka broker.
- `ProcessGroupName`– This is used as a dimension for JVM metrics collected for brokers. Provide the same value as you provide for `ClusterName`. This enables viewing the JVM metrics of the same Kafka broker group as the broker metrics in the solution dashboard.

- `ProducerGroupName`– This is used as a dimension for producer metrics collected. Provide a meaningful name that represents the group of producer instances. For this value, you can specify your producer application or service that you want to use for a combined view of producer metrics in the solution dashboard.
- `ConsumerGroupName`– This is used as a dimension for consumer metrics collected. Provide a meaningful name that represents the group of consumer instances This is not the same as the consumer group concept in Kafka. This is just a grouping dimension where you can specify your consumer application or service that you want to use for a combined view of consumer metrics in the solution dashboard

For example, if you have two Kafka clusters running in the same account, one for the `order-processing` application and another for the `inventory-management` application, you should set the `ClusterName` and `ProcessGroupName` dimensions accordingly in the agent configuration of the broker instance.

- For the `order-processing` cluster broker instances, set `ClusterName=order-processing` and `ProcessGroupName=order-processing`.
- For the `inventory-management` cluster broker instances, set `ClusterName=inventory-management` and `ProcessGroupName=inventory-management`.
- Similarly, set the `ProducerGroupName` for producer instances and `ConsumerGroupName` for consumer instances based on their respective applications.

When you correctly set the above dimensions, the solution dashboard will automatically group the metrics based on the `ClusterName`, `ProducerGroupName`, and `ConsumerGroupName` dimensions. The dashboard will include dropdown options to select and view metrics for specific clusters and groups, allowing you to monitor the performance of individual clusters and groups separately.

Be sure to deploy the relevant agent configuration to the correct EC2 instances. Each configuration will be stored as a separate Parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

The following instructions describe the situation where the producer, consumer, and broker roles are deployed to separate EC2 instances, without any overlap. If you are running multiple Kafka roles on the same EC2 instances, see Configure the agent for multiple Kafka roles on the same instance for more information.

Agent configuration for Kafka broker agents

Use the following CloudWatch agent configuration on EC2 instances where Kafka broker agents are deployed. Replace *ClusterName* with the name of the cluster to use to group these metrics for a unified view. The value you specify for *ClusterName* is used as both the ClusterName dimension and the ProcessGroupName dimension. Replace *port-number* with the JMX port of your Kafka server. If JMX was enabled with password authentication or SSL for remote access, see Collect Java Management Extensions (JMX) metrics for information about setting up TLS or authorization as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka": {
            "measurement": [
              "kafka.request.time.avg",
              "kafka.request.failed",
              "kafka.request.count",
              "kafka.purgatory.size",
              "kafka.partition.under_replicated",
              "kafka.partition.offline",
              "kafka.network.io",
              "kafka.leader.election.rate",
              "kafka.isr.operation.count"
            ]
          },
          "append_dimensions": {
            "ClusterName": "ClusterName"
          }
        },
```

```
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "ClusterName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      },
      "swap": {
        "measurement": [
          "used_percent"
        ]
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_time_wait"
        ]
      }
    }
  }
```

```
}
```

Agent configuration for Kafka producers

Use the following CloudWatch agent configuration on Amazon EC2 instances where Kafka producers are deployed. Replace *ProducerGroupName* with the name of the application or group that you want to use to group your metrics for a unified view. Replace *port-number* with the JMX port of your Kafka producer application.

The solution doesn't enable JVM metrics for Kafka producers because the solution dashboard doesn't display JVM metrics related to JVM for producers. You can customize the agent configuration to emit JVM metrics as well, however, JVM metrics related to producers are not visible on the solution dashboard.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-producer": {
            "measurement": [
              "kafka.producer.request-rate",
              "kafka.producer.byte-rate",
              "kafka.producer.request-latency-avg",
              "kafka.producer.response-rate",
              "kafka.producer.record-error-rate",
              "kafka.producer.record-send-rate"
            ]
          },
          "append_dimensions": {
            "ProducerGroupName": "ProducerGroupName"
          }
        }
      ]
    }
  }
}
```

Agent configuration for Kafka consumers

Use the following CloudWatch agent configuration on EC2 instances where Kafka consumers are running. Replace *ConsumerGroupName* with the name of the application or group to use to group these metrics for a unified view. Replace *port-number* with the JMX port of your Kafka consumer application.

The solution doesn't enable JVM metrics for Kafka consumers because the solution dashboard doesn't display JVM metrics related to JVM for consumers. You can customize the agent configuration to emit JVM metrics as well, however JVM metrics related to consumer are not visible on the solution dashboard.

```
{
  "metrics": {
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-consumer": {
            "measurement": [
              "kafka.consumer.fetch-rate",
              "kafka.consumer.total.bytes-consumed-rate",
              "kafka.consumer.records-consumed-rate",
              "kafka.consumer.bytes-consumed-rate",
              "kafka.consumer.records-lag-max"
            ]
          },
          "append_dimensions": {
            "ConsumerGroupName": "ConsumerGroupName"
          }
        }
      ]
    }
  }
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the workload is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration according to the role (broker, producer, or consumer) with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

You must repeat these steps based on whether your broker, producer, and consumer are deployed on the same EC2 instance or different instances. For example, if the Kafka broker, producer, and

consumers are getting deployed on separate instances without overlap, you must repeat these steps three times with the appropriate agent configurations for broker, producer, and consumer EC2 instances.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.

    a. Choose **Create parameter**.

    b. Provide a name for the parameter that will store your CloudWatch agent configuration, such as **`AmazonCloudWatch-Kafka-Producer-Configuration`** for producers, **`AmazonCloudWatch-Kafka-Consumer-Configuration`** for consumers, or **`AmazonCloudWatch-Kafka-Broker-Configuration`** for brokers. If you have multiple

Kafka roles on a single EC2, name the roles accordingly for easier identification. This value will later be used to distribute this configuration to the agent running on your EC2 instance.

c. For **Parameter tier**, choose **Standard**.

d. For **Type**, choose **String**.

e. For **Data type**, choose **text**.

f. In the **Value** box, paste the full text of the CloudWatch agent configuration. Be sure to select the JSON block for the Kafka role that this instance is hosting. Refer to the configuration provided in Agent configuration for Kafka broker agents, Agent configuration for Kafka producers, and Agent configuration for Kafka consumers when storing the configuration for broker, producer, and consumer respectively. If you are running multiple Kafka roles on the same EC2 instance, be sure to merge the configuration if required as described in Configure the agent for multiple Kafka roles on the same instance on the same instance

g. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use AWS CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-Kafka-Broker-Configuration`** for brokers, **`AmazonCloudWatch-Kafka-Producer-`**

`Configuration` for producers, and **`AmazonCloudWatch-Kafka-Consumer-Configuration`** for consumers.

b. To select the target instances, you have two options.

i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.

Note

After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:

1. If the Systems Manager parameter is deleted, the agent will stop.
2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Make sure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

If everything is set up correctly, then you should see the Kafka metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that Kafka metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `kafka.partition.offline` for brokers, `kafka.consumer.fetch.rate` for consumers, or `kafka.producer.request-rate` for producers. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the Kafka solution dashboard

This dashboard displays the newly emitted metrics for both Kafka and the underlying JVM. This dashboard provides a top contributor view for the health of your Kafka workload, across producers, brokers, and consumers. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for AWS services, see Viewing a CloudWatch dashboard for a single AWS service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your JVM and Kafka metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard through CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=ApacheKafkaOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`KafkaDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Kafka_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Kafka workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`KafkaDashboardStack`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`KafkaDashboard-us-east-1`**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.
7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Kafka_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the Kafka dashboard

Here are a few tasks that you can try out with the new Kafka dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a Kafka cluster. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Using the dropdown lists

The dashboard provides dropdown lists at the top that you can use to filter and select the specific Kafka cluster, producer, and consumer groups that you want to monitor.

- To display metrics for a specific Kafka cluster, select that cluster name in the **Kafka Cluster** dropdown list.
- To display metrics for a specific Kafka producer group, select that producer group name in the **Kafka Producer** dropdown list.
- To display metrics for a specific Kafka consumer group, select that consumer group name in the **Kafka Consumer Group** dropdown list.

Verify cluster health

From the **Cluster Overview** section, find the **Partitions Under Replicated** and **In-Sync Replicas** widgets. These should ideally be zero or a small number. A large value for any of these metrics could indicate issues with the Kafka cluster that need investigation.

Investigate broker performance

In the **Brokers** section, find the **Failed Fetch Requests** and **Failed Producer Requests** widgets. These show the number of failed requests for fetch and produce operations, respectively. High failure rates could indicate issues with the brokers or network connectivity that require further investigation.

Monitor producer performance

In the **Producer Group Overview** section, find the **Average Request Rate**, **Average Request Latency**, and **Average Record Send/Error Rate** widgets. These will give you an overview of how the producers in the selected group are performing. You can also drill down to view metrics for specific producers and topics in the **Producers** section.

Monitor consumer lag

In the **Consumer Group Overview** section, find the **Consumer Lag** widget. This shows how far behind the consumers are in processing messages from the latest offsets in the partitions they are subscribed to. Ideally, the consumer lag should be low or zero. A high consumer lag could indicate that the consumers are unable to keep up with the rate of data production, leading to potential data loss or delays in processing. You can also drill down to view metrics for specific consumers and topics in the **Consumers** section.

Configure the agent for multiple Kafka roles on the same instance

The individual configurations for Kafka roles listed in CloudWatch agent configuration for this solution apply only when the producer, consumer, and broker roles are deployed on separate EC2 instances, without any overlap. If you are running multiple Kafka roles on the same Amazon EC2 instances, you have two options:

- Create a single agent configuration file which lists and configures all metrics for all the Kafka roles deployed on that instance. If you are going to use Systems Manager to manage agent configuration, this is the preferred option.

If you choose this option and the multiple Kafka roles are part of the same JVM process, you must specify the same endpoint for each Kafka role in the agent configuration. If the multiple Kafka roles are part of different JVM processes, the endpoint for each role can be different depending on the JMX port set for that process.

- Create separate agent configuration files for each Kafka role, and configure the agent to apply both configuration files. For instructions for applying multiple configuration files, see Creating multiple CloudWatch agent configuration files.

The following example shows a CloudWatch agent configuration where the producer and consumer roles are running on one instance as part of the same JVM process. In this case, the port number must be the same in both the producer and consumer parts of the configuration below. If instead the two roles were running as part of different JVM processes, you could specify different port numbers for each, according to the JMX port of each individual JVM process.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "kafka-producer": {
            "measurement": [
              "kafka.producer.request-rate",
              "kafka.producer.byte-rate",
              "kafka.producer.request-latency-avg",
              "kafka.producer.response-rate",
              "kafka.producer.record-error-rate",
              "kafka.producer.record-send-rate"
            ]
          },
          "append_dimensions": {
            "ProducerGroupName": "ProducerGroupName"
          }
        },
        {
          "endpoint": "localhost:port-number",
```

```
          "kafka-consumer": {
            "measurement": [
              "kafka.consumer.fetch-rate",
              "kafka.consumer.total.bytes-consumed-rate",
              "kafka.consumer.records-consumed-rate",
              "kafka.consumer.bytes-consumed-rate",
              "kafka.consumer.records-lag-max"
            ]
          },
          "append_dimensions": {
            "ConsumerGroupName": "ConsumerGroupName"
          }
        }
      ]
    }
  }
}
```

CloudWatch solution: Tomcat workload on Amazon EC2

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for Tomcat server running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions.

Topics

- Requirements
- Benefits
- Costs
- CloudWatch agent configuration for this solution
- Deploy the agent for your solution
- Create the Tomcat solution dashboard

Requirements

This solution is relevant for the following conditions:

- Supported versions: Tomcat versions 9, 10.1, and 11 (beta)

- Compute: Amazon EC2
- Supports up to 500 EC2 instances across all Tomcat workloads in a given AWS Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux
> - Manually installing and uninstalling SSM Agent on EC2 instances for macOS
> - Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers Tomcat server monitoring, providing valuable insights for the following use cases:

- Detect Tomcat server errors and performance issues.
- Monitor network traffic for data transfer problems.
- Track thread usage and active user sessions.
- Analyze underlying JVM performance for Tomcat server.

Below are the key advantages of the solution:

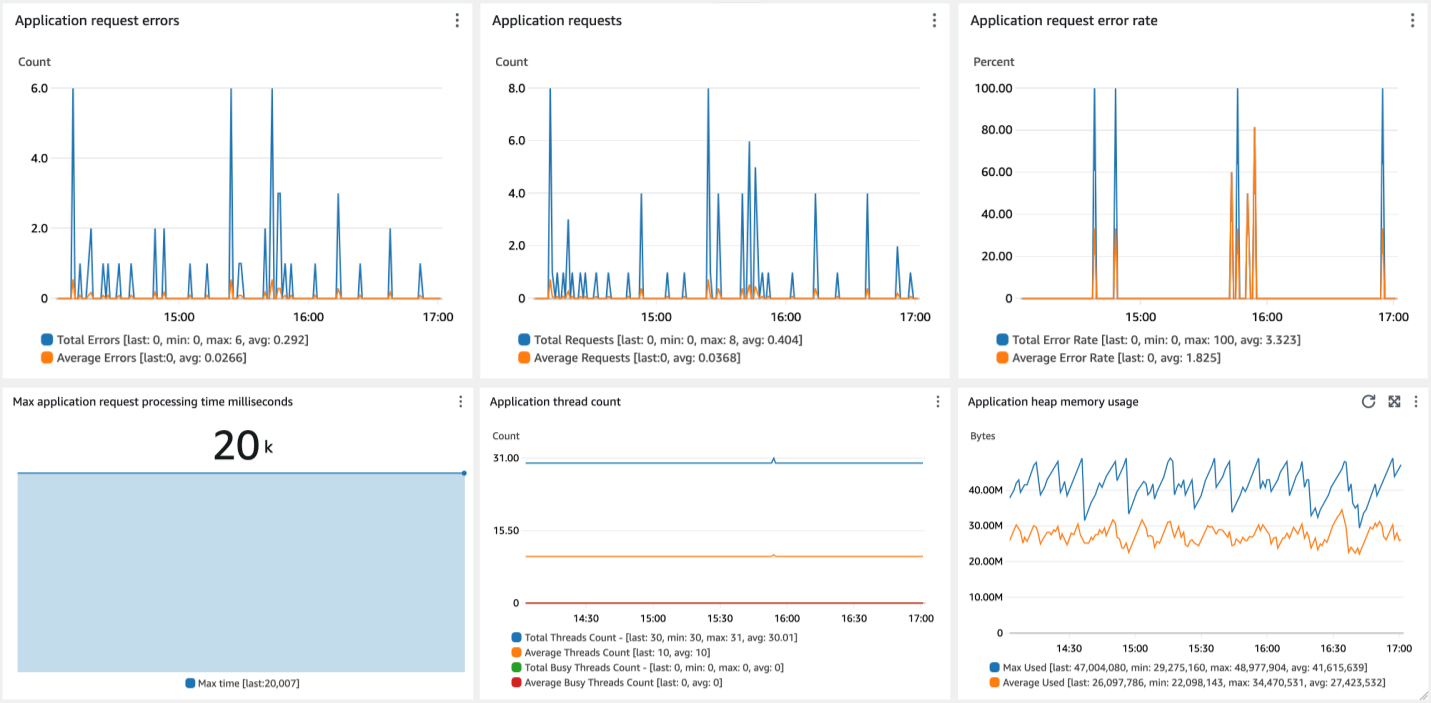
- Automates metric collection for Apache Tomcat and the underlying JVM using CloudWatch agent configuration, eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for Apache Tomcat and JVM metrics. The dashboard will automatically handle metrics from new Tomcat EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to group the metrics into logical applications for easier focus and management.

The following image is an example of the dashboard for this solution.

App overview

This part of the dashboard gives you a high-level summary and overall picture of your selected Tomcat App and selected duration using metrics aggregated from all instances in your cluster.

Note: View data from different Tomcat Apps using the `Tomcat App` dropdown at the top of the dashboard.



Breakdown by server

The following sections of the dashboard give you a more detailed look at the top contributing servers for different Tomcat App metrics.

Note: View data from different Tomcat Apps using the `Tomcat App` dropdown at the top of the dashboard.

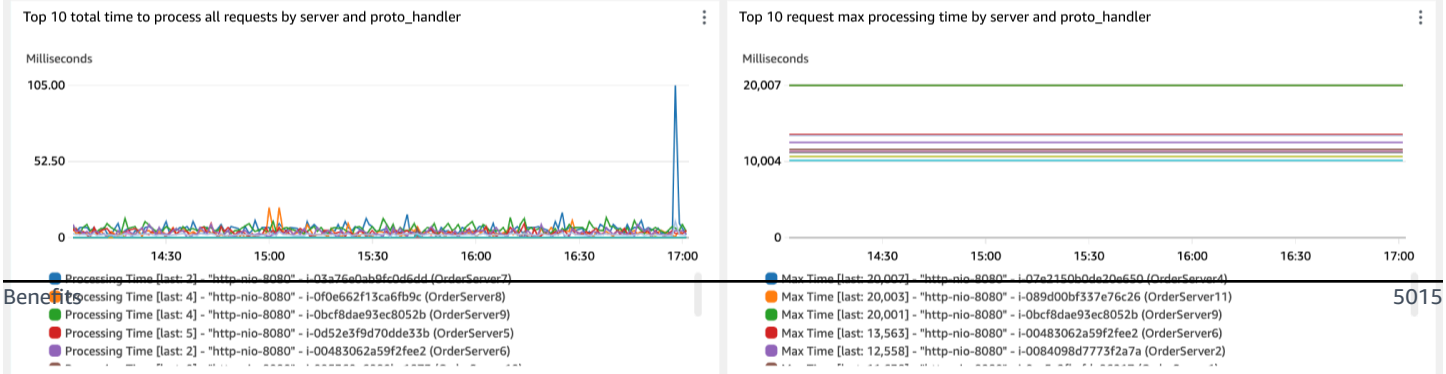
Requests

This section displays the number of errors encountered during request processing and total number of requests processed by Tomcat. Use the error count to identify and troubleshoot potential issues. Additionally, the count metric helps you understand the load on your server.



Request processing time

This section displays the total time spent processing all requests and the maximum time spent by Tomcat processing a single request. These metrics help identify performance bottlenecks and provide insights into the overall request load on your server.



Sessions and threads

This section provides insights into the thread and session management of your Tomcat server. It displays the number of busy threads and the total current thread count, allowing you to monitor thread utilization and

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.
  - Each Tomcat host configured for the solution publishes a total of 27 metrics plus one metric (`disk_used_percent`) for which the metric count depends on number of disk paths for that host.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. In the **Metrics** section, for **Number of metrics**, enter **`(27 + average number of disk paths per EC2 host) * number of EC2 instances configured for this solution`**.
3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.

   By default, the solution performs one **PutMetricData** operation each minute for each EC2 host.
4. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
5. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces using the CloudWatch agent.

The agent configuration in this solution collects the foundational metrics for Tomcat, JVM, and EC2. The CloudWatch agent can be configured to collect more JVM metrics than the dashboard displays by default. For a list of all Tomcat metrics that you can collect, see Collect Tomcat metrics. For a list of all JVM metrics that you can collect, see Collect JVM metrics. For a list of Amazon EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances.

Expose JMX ports for the Tomcat server

The CloudWatch agent relies on JMX to collect the metrics related to the Tomcat server and JVM process. To make this possible, you must expose the JMX port from your servers. To enable a JMX port for monitoring and management, you would set system properties for your Tomcat servers. You can use the environment variable `CATALINA_OPTS` to set the required system properties for Tomcat. Review the startup scripts and configuration files of your Tomcat server on the best place to set the environment variable. Be sure that you specify an unused port number. You will need to restart the server after the change.

```
export CATALINA_OPTS="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=<<port-number>> -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.authenticate=false"
```

This example sets up unauthenticated JMX. If your security policies/requirements require you to enable JMX with password authentication or SSL for remote access, refer to the JMX documentation to set the required property.

To verify the JMX port, run `ps aux | grep jmxremote.port`. The results should show that the JMX port was set on the JVM processes.

Agent configuration for Tomcat solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in Deploy the agent for your solution. The following information is intended to help you understand how to customize the agent configuration for your environment.

You must customize some parts of the following agent configuration for your environment:

- The JMX port number is the port number that you configured in the previous section of this documentation. The port number is in the `endpoint` line in the configuration.
- `AppName` – This is used as a dimension for the Tomcat application metrics collected. Provide a meaningful name that represents the grouping for the instances that run the Tomcat application.
- `ProcessGroupName` – This is used as a dimension for JVM metrics collected for Tomcat hosts. Provide the value which is the same as `AppName` above. This is to enable viewing the JVM metrics of the same Tomcat app group as server metrics under the solution dashboard.

For example, if you have two Tomcat apps running in the same AWS account, one for the `billing-system` application and another for the `order-system` application, you can set the `AppName` and `ProcessGroupName` dimensions accordingly in the agent configuration of each instance.

- For the `billing-system` application instances, set `AppName=billing-system` and `ProcessGroupName=billing-system`.
- For the `order-system` application instances, set `AppName=order-system` and `ProcessGroupName=order-system`.

When you follow these guidelines, the solution will automatically group the metrics based on the `AppName` and `ProcessGroupName` dimensions. The dashboard will include dropdown options to select and view metrics for a specific Tomcat application, allowing you to monitor the performance of individual applications separately.

Agent configuration for Tomcat hosts

Use the following CloudWatch agent configuration on EC2 instances where your Tomcat applications are deployed. Configuration will be stored as a parameter in SSM's Parameter Store, as detailed later in Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store.

Replace *AppName* with a meaningful name that represents the Tomcat application the instances are part of. Replace *port-number* with the JMX port of your Tomcat server. If JMX was enabled with password authentication or SSL for remote access, see Collect Java Management Extensions (JMX) metrics for information about setting up TLS or authorization in agent configuration as required.

The EC2 metrics shown in this configuration (configuration shown outside the JMX block) only work for Linux and macOS instances. If you are using Windows instances, you can choose to omit these metrics in the configuration. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

```
{
  "metrics": {
    "namespace": "CWAgent",
    "append_dimensions": {
      "InstanceId": "${aws:InstanceId}"
    },
    "metrics_collected": {
      "jmx": [
        {
          "endpoint": "localhost:port-number",
          "tomcat": {
            "measurement": [
              "tomcat.sessions",
              "tomcat.errors",
              "tomcat.processing_time",
              "tomcat.traffic",
              "tomcat.max_time",
              "tomcat.request_count",
              "tomcat.threads"
            ]
          },
          "append_dimensions": {
            "AppName": "AppName"
          }
        },
```

```
        {
          "endpoint": "localhost:port-number",
          "jvm": {
            "measurement": [
              "jvm.classes.loaded",
              "jvm.gc.collections.count",
              "jvm.gc.collections.elapsed",
              "jvm.memory.heap.committed",
              "jvm.memory.heap.max",
              "jvm.memory.heap.used",
              "jvm.memory.nonheap.committed",
              "jvm.memory.nonheap.max",
              "jvm.memory.nonheap.used",
              "jvm.threads.count"
            ]
          },
          "append_dimensions": {
            "ProcessGroupName": "AppName"
          }
        }
      ],
      "disk": {
        "measurement": [
          "used_percent"
        ]
      },
      "mem": {
        "measurement": [
          "used_percent"
        ]
      },
      "swap": {
        "measurement": [
          "used_percent"
        ]
      },
      "netstat": {
        "measurement": [
          "tcp_established",
          "tcp_time_wait"
        ]
      }
    }
  }
```

```
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts where the JVM application is deployed and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the agent configuration of JVM with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in this solution are the minimum required for the recommended dashboard.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. Follow the steps in Launch an instance with an IAM role to attach a role while launching a new EC2 instance. To attach a role to an existing EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. From the navigation pane, choose **Application Management**, **Parameter Store**.
3. Follow these steps to create a new parameter for the configuration.
    a. Choose **Create parameter**.
    b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-Tomcat-Configuration`**.
    c. (Optional) In the **Description** box, type a description for the parameter.
    d. For **Parameter tier**, choose **Standard**.
    e. For **Type**, choose **String**.
    f. For **Data type**, choose **text**.

g. In the **Value** box, paste the corresponding JSON block that was listed in Agent configuration for Tomcat hosts. Be sure to customize the grouping dimension value and port number as described.

h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use AWS CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-Tomcat-Configuration`**.

   b. To select the target instances, you have two options.

      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.

5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json.

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.
> 2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
> 3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in Step 1: Make sure the target EC2 instances have the required IAM permissions.
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

If everything is set up correctly, then you should see the Tomcat metrics being published to CloudWatch. You can check the CloudWatch console to verify they are being published.

To verify that Tomcat metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `tomcat.errors`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the Tomcat solution dashboard

This dashboard displays the newly emitted metrics, showing both the Tomcat application server and the underlying JVM. This dashboard provides a top contributor view for the health of your Tomcat workload. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance. The dashboard also includes an overview of the cluster by aggregating and presenting metrics across all instances, providing a high-level summary of the overall health and operational state of the cluster.

The solution dashboard doesn't display EC2 metrics. To view EC2 metrics, you'll need to use the EC2 automatic dashboard to see EC2 vended metrics and use the EC2 console dashboard to see EC2 metrics that are collected by the CloudWatch agent. For more information about automatic dashboards for AWS services, see Viewing a CloudWatch dashboard for a single AWS service.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your Tomcat metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard through CloudWatch Console

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=ApacheTomcatOnEc2&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`TomcatDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Tomcat_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.
2. Verify that the selected Region on the console is the Region where the Tomcat workload is running.
3. For **Stack name**, enter a name to identity this stack, such as **`TomcatDashboard-us-east-1`**.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.
5. To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`TomcatDashboard-us-east-1`**.
6. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
7. Review the settings, then choose **Create stack**.
8. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/Tomcat_EC2/CloudWatch/CFN/v1.0.0/dashboard-template-1.0.0.json.

> **Note**
>
> Solution dashboards currently display garbage collection-related metrics only for the G1 Garbage Collector, which is the default collector for the latest Java versions. If you are using a different garbage collection algorithm, the widgets pertaining to garbage collection are empty. However, you can customize these widgets by changing the dashboard CloudFormation template and applying the appropriate garbage collection type to the name dimension of the garbage collection-related metrics. For

example, if you are using parallel garbage collection, change the **`name=\"G1 Young Generation\"`** to **`name=\"Parallel GC\"`** of the garbage collection count metric `jvm.gc.collections.count`.

Get started with the Tomcat monitoring dashboard

Here are a few tasks that you can try out with the new Tomcat dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor a Tomcat application. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Using the dropdown list

The dashboard provides a dropdown list at the top that you can use to filter and select the specific Tomcat application that you want to monitor. To display metrics for a specific Tomcat application, select that application name in the **Tomcat App** dropdown list.

Verify application health

From the **App Overview** section, find the **Requests**, **Errors**, and **Error Rate** widgets. These provide a high-level summary of the application's request handling performance. Look for any abnormally high error counts or rates, which could indicate issues that need investigation.

Monitor request processing

In the **Request Processing Time** section, find the **Max Time** and **Total Time to Process All Requests** widgets. These metrics help you identify potential performance bottlenecks in request processing. Look for any servers with significantly higher max processing times compared to others.

Analyze network traffic

In the **Network Traffic** section, find the **Sent Traffic** and **Received Traffic** widgets. These show the amount of data being sent and received by the application over the network. Unexpectedly high traffic levels could indicate potential issues with network saturation or inefficient data transfer.

Investigate thread usage

In the **Sessions and Threads** section, find the **Busy Threads Count**, **Threads Count**, and **Sessions** widgets. These metrics provide insights into the application's thread management and active user

sessions. Look for any servers with an abnormally high number of busy threads or sessions, which could indicate potential resource constraints.

CloudWatch solution: Amazon EC2 health

This solution helps you configure out-of-the-box metric collection using CloudWatch agents for workloads running on EC2 instances. Additionally, it helps you set up a pre-configured CloudWatch dashboard.

Topics

- Requirements
- Benefits
- Costs
- CloudWatch agent configuration for this solution
- Deploy the agent for your solution
- Create the EC2 Health solution dashboard
- Get started with the EC2 Health solution dashboard

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon EC2
- Platform: Linux and macOS
- Supports up to 500 EC2 instances in a given AWS Region
- Latest version of CloudWatch agent
- SSM agent installed on EC2 instance

> **Note**
>
> AWS Systems Manager (SSM agent) is pre-installed on some Amazon Machine Images (AMIs) provided by AWS and trusted third-parties. If the agent isn't installed, you can install it manually using the procedure for your operating system type.
>
> - Manually installing and uninstalling SSM Agent on EC2 instances for Linux

- Manually installing and uninstalling SSM Agent on EC2 instances for macOS
- Manually installing and uninstalling SSM Agent on EC2 instances for Windows Server

Benefits

The solution delivers EC2 server monitoring using the CloudWatch Agent, providing additional system-level metrics on top of the standard EC2 namespace metrics for the following use cases:

- Detect CPU performance issues and resource constraints.
- Monitor disk utilization and storage capacity across different disks throughout your EC2 instances.
- Track memory usage patterns and potential memory leaks.
- Analyze I/O operations and their impact on overall performance.
- Observe network traffic patterns and potential anomalies.

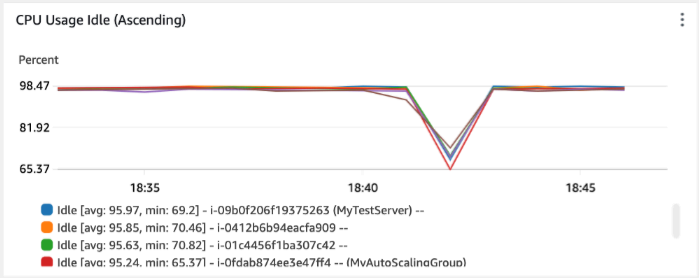
Below are the key advantages of the solution:

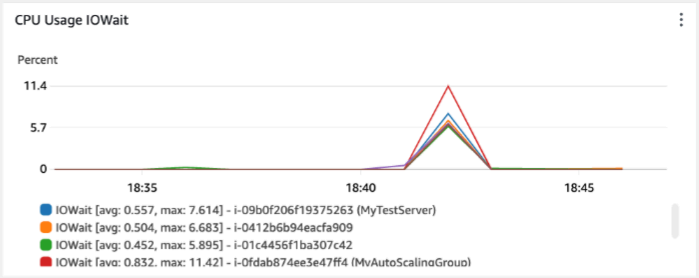
- Automates metric collection for EC2 instances eliminating manual instrumentation.
- Provides a pre-configured, consolidated CloudWatch dashboard for EC2 instance metrics. The dashboard will automatically handle metrics from new EC2 instances configured using the solution, even if those metrics don't exist when you first create the dashboard. It also allows you to observe EC2 instances managed through Auto Scaling groups.

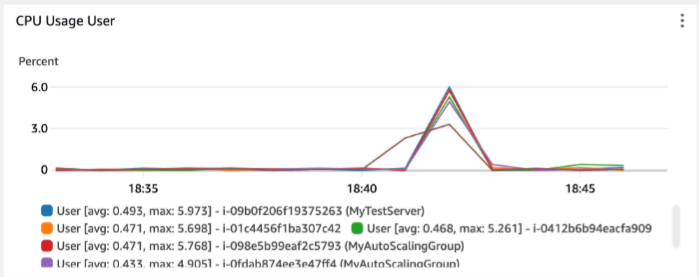
The following image is an example of the dashboard for this solution.

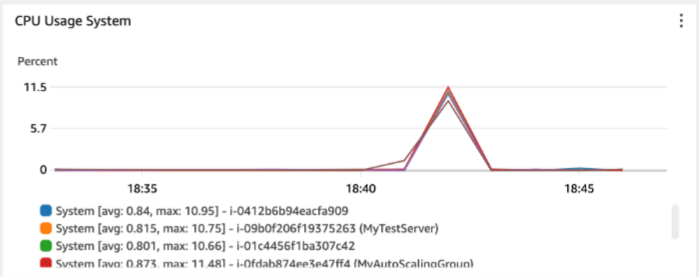
CloudWatch Agent Metrics

CPU







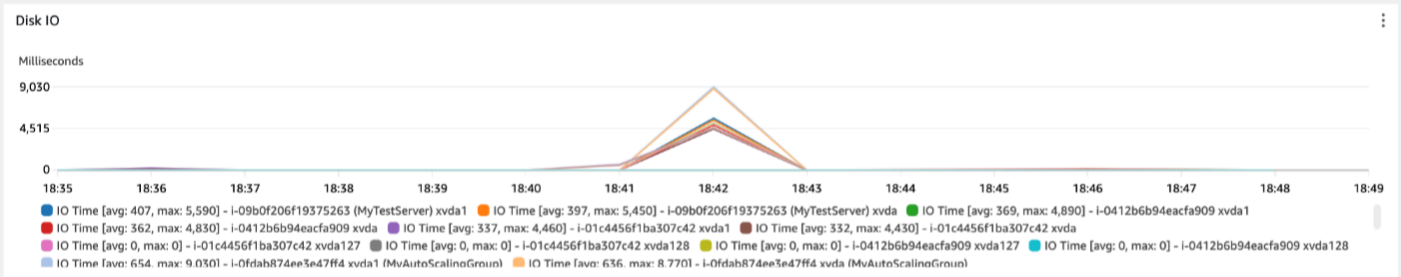


Disk

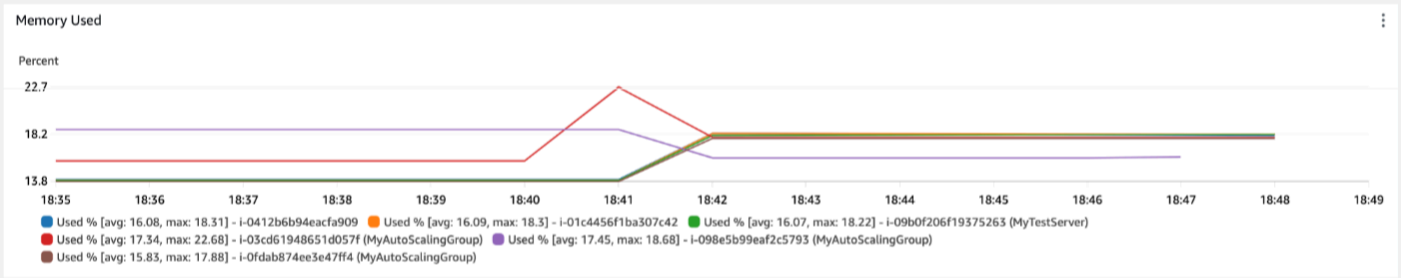




DiskIO



Memory



Swap



AWS/EC2 Vended Metrics

Costs

This solution creates and uses resources in your account. You are charged for standard usage, including the following:

- All metrics collected by the CloudWatch agent are charged as custom metrics. The number of metrics used by this solution depends on the number of EC2 hosts.

  The total number of CloudWatch agent metrics depends on the configuration of disks. Excluding disk and diskio metrics, the solution publishes six metrics. The number of disk metrics (`disk_used_percent`, `disk_inodes_free`) depend on the count of `device/fstype/path` dimensions. The diskio metric (`diskio_io_time`) depends on the count of `name` dimensions. For example, a single t2.micro with default settings as per EC2 console experience, produces a total of 22 CloudWatch agent metrics (4 CPU, 12 disk, 4 diskio, 1 memory, and 1 swap). Vended metrics like `AWS/EC2` are provided free of charge.
- One custom dashboard.
- API operations requested by the CloudWatch agent to publish the metrics. With the default configuration for this solution, the CloudWatch agent calls the **PutMetricData** once every minute. This means the **PutMetricData** API will be called `30*24*60=43,200` in a 30-day month for each EC2 host.

For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

The pricing calculator can help you estimate approximate monthly costs for using this solution.

To use the pricing calculator to estimate your monthly solution costs

1. Open the Amazon CloudWatch pricing calculator.
2. In the **Metrics** section, for **Number of metrics**, enter **`(6 + total count of disk and diskio metrics per EC2 host as described above) * number of EC2 instances configured for this solution`**.
3. In the **APIs** section, for **Number of API requests**, enter **`43200 * number of EC2 instances configured for this solution`**.
4. By default, the solution performs one **PutMetricData** operation each minute for each EC2 host.
5. In the **Dashboards and Alarms** section, for **Number of Dashboards**, enter **1**.
6. You can see your monthly estimated costs at the bottom of the pricing calculator.

CloudWatch agent configuration for this solution

The CloudWatch agent is software that runs continuously and autonomously on your servers and in containerized environments. It collects metrics, logs, and traces from your infrastructure and applications and sends them to CloudWatch and X-Ray.

For more information about the CloudWatch agent, see Collect metrics, logs, and traces with the CloudWatch agent.

The agent configuration in this solution collects a set of metrics to help you get started monitoring and observing your EC2 instances. The CloudWatch agent can be configured to collect more EC2 metrics than the dashboard displays by default. For a list of Amazon EC2 metrics, see Metrics collected by the CloudWatch agent on Linux and macOS instances. For information about metrics collected on Windows instances, see Metrics collected by the CloudWatch agent on Windows Server instances.

Agent configuration for EC2 Health solution

The metrics collected by the agent are defined in the agent configuration. The solution provides agent configurations to collect the recommended metrics with suitable dimensions for the solution's dashboard.

The steps for deploying the solution are described later in the section called “Deploy the agent for your solution”. The following information is intended to help you understand how to customize the agent configuration for your environment.

> **Note**
>
> If an EC2 instance is not part of an Auto Scaling group, the CloudWatch agent drops the `AutoScalingGroupName` dimension entirely. This behavior helps to prevent dimension names with null/empty values. Each metric widget included in the solution dashboard searches for metrics which include and exclude the `AutoScalingGroup` dimension. This helps to make sure that all EC2 instances where the solution is applied are supported by the same dashboard.

If you wish to make any modifications to the agent configuration, you must apply the same changes to the solution's accompanying dashboard. For example, if you decide to omit the ImageId

dimension, the same dimension must be removed from the metric search expression used in the dashboard widgets.

Agent configuration for EC2 Instances

Use the following CloudWatch agent configuration on Amazon EC2 instances where your workloads are deployed.

```
{
    "agent": {
      "metrics_collection_interval": 60,
      "run_as_user": "cwagent"
    },
    "metrics": {
      "append_dimensions": {
        "InstanceId": "${aws:InstanceId}",
        "InstanceType": "${aws:InstanceType}",
        "ImageId": "${aws:ImageId}",
        "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
      },
      "metrics_collected": {
        "cpu": {
          "measurement": [
            "cpu_usage_idle",
             "cpu_usage_iowait",
             "cpu_usage_user",
             "cpu_usage_system"
          ],
          "totalcpu": true
        },
        "disk": {
          "measurement": [
            "used_percent",
            "inodes_free"
          ],
          "resources": [
            "*"
          ],
          "dimensions": [
            ["device", "fstype", "path"]
          ]
        },
        "diskio": {
```

```
            "measurement": [
              "io_time"
            ],
            "resources": [
              "*"
            ]
          },
        "mem": {
          "measurement": [
            "used_percent"
          ]
        },
        "swap": {
          "measurement": [
            "used_percent"
          ]
        }
      }
    }
}
```

Deploy the agent for your solution

There are several approaches for installing the CloudWatch agent, depending on the use case. We recommend using Systems Manager for this solution. It provides a console experience and makes it simpler to manage a fleet of managed servers within a single AWS account. The instructions in this section use Systems Manager and are intended for when you don't have the CloudWatch agent running with existing configurations. You can check whether the CloudWatch agent is running by following the steps in Verify that the CloudWatch agent is running.

If you are already running the CloudWatch agent on the EC2 hosts and managing the agent configurations, you can skip the instructions in this section and follow your existing deployment mechanism to update the configuration. Be sure to merge the EC2 Health agent configuration with your existing agent configuration, and then deploy the merged configuration. If you are using Systems Manager to store and manage the configuration for the CloudWatch agent, you can merge the configuration to the existing parameter value. For more information, see Managing CloudWatch agent configuration files.

> **Note**
>
> Using Systems Manager to deploy the following CloudWatch agent configurations will replace or overwrite any existing CloudWatch agent configuration on your EC2 instances. You can modify this configuration to suit your unique environment or use case. The metrics defined in configuration are the minimum required for the dashboard provided the solution.

The deployment process includes the following steps:

- Step 1: Make sure that the target EC2 instances have the required IAM permissions.
- Step 2: Store the recommended agent configuration file in the Systems Manager Parameter Store.
- Step 3: Install the CloudWatch agent on one or more EC2 instances using an CloudFormation stack.
- Step 4: Verify the agent setup is configured properly.

Step 1: Make sure the target EC2 instances have the required IAM permissions

You must grant permission for Systems Manager to install and configure the CloudWatch agent. You must also grant permission for the CloudWatch agent to publish telemetry from your EC2 instance to CloudWatch. Make sure that the IAM role attached to the instance has the **CloudWatchAgentServerPolicy** and **AmazonSSMManagedInstanceCore** IAM policies attached.

- After the role is created, attach the role to your EC2 instances. To attach a role to an EC2 instance, follow the steps in Attach an IAM role to an instance.

Step 2: Store the recommended CloudWatch agent configuration file in Systems Manager Parameter Store

Parameter Store simplifies the installation of the CloudWatch agent on an EC2 instance by securely storing and managing configuration parameters, eliminating the need for hard-coded values. This ensures a more secure and flexible deployment process, enabling centralized management and easier updates to configurations across multiple instances.

Use the following steps to store the recommended CloudWatch agent configuration file as a parameter in Parameter Store.

To create the CloudWatch agent configuration file as a parameter

1. Open the AWS Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. From the navigation pane, choose **Application Management**, **Parameter Store**.
4. Follow these steps to create a new parameter for the configuration.
   a. Choose **Create parameter**.
   b. In the **Name** box, enter a name that you'll use to reference the CloudWatch agent configuration file in later steps. For example, **`AmazonCloudWatch-EC2Health-Configuration`**.
   c. (Optional) In the **Description** box, type a description for the parameter.
   d. For **Parameter tier**, choose **Standard**.
   e. For **Type**, choose **String**.
   f. For **Data type**, choose **text**.
   g. In the **Value** box, paste the agent configuration JSON provided earlier in this document.
   h. Choose **Create parameter**.

Step 3: Install the CloudWatch agent and apply the configuration using an CloudFormation template

You can use CloudFormation to install the agent and configure it to use the CloudWatch agent configuration that you created in the previous steps.

To install and configure the CloudWatch agent for this solution

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .

2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. For **Stack name**, enter a name to identity this stack, such as **`CWAgentInstallationStack`**.
4. In the **Parameters** section, specify the following:

   a. For **CloudWatchAgentConfigSSM**, enter the name of the Systems Manager parameter for the agent configuration that you created earlier, such as **`AmazonCloudWatch-EC2Health-Configuration`**.

   b. To select the target instances, you have two options.

      i. For **InstanceIds**, specify a comma-delimited list of instance IDs list of instance IDs where you want to install the CloudWatch agent with this configuration. You can list a single instance or several instances.

      ii. If you are deploying at scale, you can specify the **TagKey** and the corresponding **TagValue** to target all EC2 instances with this tag and value. If you specify a **TagKey**, you must specify a corresponding **TagValue**. (For an Auto Scaling group, specify **`aws:autoscaling:groupName`** for the **TagKey** and specify the Auto Scaling group name for the **TagValue** to deploy to all instances within the Auto Scaling group.)

   If you specify both the **InstanceIds** and the **TagKeys** parameters, the **InstanceIds** will take precedence and the tags will be ignored.
5. Review the settings, then choose **Create stack**.

If you want to edit the template file first to customize it, choose the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use the following link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/CloudWatchAgent/CFN/v1.0.0/cw-agent-installation-template-1.0.0.json .

> **Note**
>
> After this step is completed, this Systems Manager parameter will be associated with the CloudWatch agents running in the targeted instances. This means that:
>
> 1. If the Systems Manager parameter is deleted, the agent will stop.

2. If the Systems Manager parameter is edited, the configuration changes will automatically apply to the agent at the scheduled frequency which is 30 days by default.
3. If you want to immediately apply changes to this Systems Manager parameter, you must run this step again. For more information about associations, see Working with associations in AWS Systems Manager.

Step 4: Verify the agent setup is configured properly

You can verify whether the CloudWatch agent is installed by following the steps in Verify that the CloudWatch agent is running. If the CloudWatch agent is not installed and running, make sure you have set up everything correctly.

- Be sure you have attached a role with correct permissions for the EC2 instance as described in the section called "Step 1: Make sure the target EC2 instances have the required IAM permissions".
- Be sure you have correctly configured the JSON for the Systems Manager parameter. Follow the steps in Troubleshooting installation of the CloudWatch agent with CloudFormation.

To verify that EC2 health metrics are being published to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, **Classic metrics**.
3. Make sure you've selected the Region where you deployed the solution, and choose **Custom namespaces**, **CWAgent**.
4. Search for the metrics mentioned in the agent configuration section of this document, such as `mem_used_percent`. If you see results for these metrics, then the metrics are being published to CloudWatch.

Create the EC2 Health solution dashboard

This dashboard displays the newly emitted metrics, showing the EC2 Health metrics. This dashboard provides a top contributor view for the health of your EC2 instances in a single region. The top contributor view displays the top 10 per metric widget. This allows you to identify outliers at a glance.

To create the dashboard, you can use the following options:

- Use CloudWatch console to create the dashboard.
- Use AWS CloudFormation console to deploy the dashboard.
- Download the AWS CloudFormation infrastructure as code and integrate it as part of your continuous integration (CI) automation.

By using the CloudWatch console to create a dashboard, you can preview the dashboard before actually creating and being charged.

> **Note**
>
> The dashboard created with CloudFormation in this solution displays metrics from the Region where the solution is deployed. Be sure to create the CloudFormation stack in the Region where your EC2 metrics are published.
> If you've specified a custom namespace other than `CWAgent` in the CloudWatch agent configuration, you'll have to change the CloudFormation template for the dashboard to replace `CWAgent` with the customized namespace you are using.

To create the dashboard through CloudWatch Console

1. Open the CloudWatch Console **Create Dashboard** using this link: https://console.aws.amazon.com/cloudwatch/home?#dashboards?dashboardTemplate=Ec2LinuxMacOsHealth&referrer=os-catalog .
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. Enter the name of the dashboard, then choose **Create Dashboard**.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`EC2HealthDashboard-us-east-1`**.
4. Preview the dashboard and choose **Save** to create the dashboard.

To create the dashboard through CloudFormation

1. Open the CloudFormation **Quick create stack** wizard using this link: https://console.aws.amazon.com/cloudformation/home?#/stacks/quickcreate?templateURL=https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/EC2_Health/CloudWatch/CFN/v1.0.0/dashboard-template-linux-macos-1.0.0.json .
2. Verify that the selected Region on the console is the Region where the EC2 instances are running.
3. For **Stack name**, enter a name to identity this stack, such as `EC2HealthDashboardStack`.
4. In the **Parameters** section, specify the name of the dashboard under the **DashboardName** parameter.

   To easily differentiate this dashboard from similar dashboards in other Regions, we recommend including the Region name in the dashboard name, such as **`EC2HealthDashboard-us-east-1`**.
5. Acknowledge access capabilities for transforms under **Capabilities and transforms**. Note that CloudFormation doesn't add any IAM resources.
6. Review the settings, then choose **Create stack**.
7. After the stack status is **CREATE_COMPLETE**, choose the **Resources** tab under the created stack and then choose the link under **Physical ID** to go to the dashboard. You can also access the dashboard in the CloudWatch console by choosing **Dashboards** in the left navigation pane of the console, and finding the dashboard name under **Custom Dashboards**.

If you want to edit the template file to customize it for any purpose, you can use the **Upload a template file** option under **Create Stack Wizard** to upload the edited template. For more information, see Creating a stack on CloudFormation console. You can use this link to download the template: https://aws-observability-solutions-prod-us-east-1.s3.us-east-1.amazonaws.com/EC2_Health/CloudWatch/CFN/v1.0.0/dashboard-template-linux-macos-1.0.0.json

Get started with the EC2 Health solution dashboard

Here are a few tasks that you can try out with the new EC2 monitoring dashboard. These tasks allow you to validate that the dashboard is working correctly and provide you some hands-on experience using it to monitor EC2 instances. As you try these out, you'll get familiar with navigating the dashboard and interpreting the visualized metrics.

Monitor the various CPU utilization metrics

In the **CPU** section, examine the array of CPU usage metrics. These provide insight into how CPU resources are being utilized across different activities like user processes, system tasks, and I/O operations. Look for instances with consistently high utilization or unusual patterns, which might indicate the need for scaling or optimization.

Analyze disk utilization across different devices

Navigate to the **Disk** section to find the storage usage and inode availability metrics. These help you identify instances that are running low on storage space or file system resources. Pay attention to any instances approaching high disk usage levels, as this could lead to performance issues or service disruptions.

Investigate memory utilization patterns

In the **Memory** section, observe the graph which plots memory utilization over time. This shows how much of the available memory is being used by each instance. Look for patterns or spikes in memory usage that might correlate with specific times or events. High memory utilization could indicate the need for instance resizing or application optimization.

Correlate patterns across core utilization metrics

Compare and watch out for related utilization patterns. For example, a workload running a log rotation process could present regular increases in **CPU** and **memory** utilization, followed by a decrease in **disk** utilization.

Inspect network activity

In the **Network** section, examine the inbound and outbound network traffic metrics, both in terms of data volume and packet count. These give you insight into the network activity for your EC2 instances. Look out for both regular or anomalous spikes in network traffic, or imbalances between inbound and outbound data.

Amazon CloudWatch solution: Prometheus metric collection on Amazon Amazon EC2

This solution helps you collect Prometheus-compatible metrics from your Amazon EC2 instances without running or scaling your own collection infrastructure. You run one or more Prometheus exporters on your instances, and a Amazon CloudWatch managed Prometheus collector

discovers the targets in your VPC, scrapes their `/metrics` endpoints, and delivers the metrics to CloudWatch. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions. For more information about managed collectors, see Amazon CloudWatch managed Prometheus collectors.

Topics

- Requirements
- Enable Prometheus exporters
- Step 1: Set up the scrape configuration
- Step 2: Set up the scraper
- Validate metrics collection
- Build a custom dashboard
- Costs

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon EC2 instances that run one or more Prometheus exporters.
- Amazon VPC with DNS enabled, and at least two subnets in different Availability Zones for the collector.
- A security group that allows the collector to reach the exporter ports on your instances.

Enable Prometheus exporters

A Prometheus exporter is a process that exposes metrics in Prometheus exposition format on an HTTP `/metrics` endpoint. You run an exporter that matches the workload you want to monitor. The following exporters are commonly used on Amazon EC2:

- **Node Exporter** — Host-level infrastructure metrics such as CPU, memory, disk, and network (default port 9100). For more information, see the Node Exporter repository on GitHub.
- **JMX Exporter** — JVM and Java application metrics for Java workloads. For more information, see the JMX Exporter repository on GitHub.
- **NGINX Prometheus Exporter** — NGINX web server and reverse proxy metrics. For more information, see the NGINX Prometheus Exporter repository on GitHub.

- **DCGM Exporter** — NVIDIA GPU metrics for GPU workloads such as machine learning training and inference (default port 9400). For more information, see the DCGM Exporter repository on GitHub.
- **HAProxy Exporter** — HAProxy load balancer metrics. For more information, see the HAProxy Exporter repository on GitHub.
- **Apache Exporter** — Apache HTTP Server metrics. For more information, see the Apache Exporter repository on GitHub.

Install and run each exporter on your instances by following its official documentation. Note the port that each exporter listens on, because you reference these ports in your scrape configuration. For a complete catalog of available exporters, see Exporters and integrations in the Prometheus documentation.

After you start an exporter, confirm that it exposes metrics by querying its endpoint from the instance:

```
curl http://localhost:port-number/metrics
```

Step 1: Set up the scrape configuration

The following example uses `static_configs` with the instance private DNS names to scrape a sample application (port 8080) and Node Exporter (port 9100) on your Amazon EC2 instances. The `relabel_configs` section adds consistent labels to your metrics so that you can query and filter across services. For the full set of supported configuration options, see Scraper configuration.

```
global:
  scrape_interval: 30s
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'ec2-metrics'
    static_configs:
      - targets:
          - ip-ip-address.us-west-2.compute.internal:8080
    metrics_path: '/metrics'
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
```

```
        replacement: 'ec2-metrics-instance'
      - target_label: service
        replacement: 'ec2-metrics'
      - target_label: environment
        replacement: 'dev'
    metric_relabel_configs:
      - source_labels: [__name__]
        regex: '.*'
        action: keep

- job_name: ec2-node-exporter
  static_configs:
    - targets:
        - ip-ip-address.us-west-2.compute.internal:9100
```

Step 2: Set up the scraper

Create a VPC-connected managed collector that scrapes your instances and delivers metrics to your CloudWatch dataset. Provide the subnets and security group that allow the collector to reach your exporter ports:

You can use GetDefaultScraperConfiguration to retrieve a general-purpose scraper configuration, or provide your own.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "ec2-payment-service-scraper",
  "source": {
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  },
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
```

```
    }
  },
  "scrapeConfiguration": {
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "ec2-payment-service-scraper" \
  --source '{
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  }' \
  --scrape-configuration configurationBlob=$(cat ec2-scraper.yml | base64 -w 0) \
  --destination '{
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  }'
```

When you create a managed collector, Amazon Managed Service for Prometheus automatically creates a service-linked role that grants the collector permission to access your VPC resources and write to your CloudWatch dataset.

Validate metrics collection

Within minutes, metrics begin flowing to your CloudWatch dataset. To confirm that your metrics arrive, run an ad-hoc query in CloudWatch using Query Studio. For example, the following query returns the metrics for the `ec2-metrics` job:

```
{job="ec2-metrics"}
```

Build a custom dashboard

After the collector begins delivering your Amazon EC2 metrics to CloudWatch, you can build a custom CloudWatch dashboard to visualize them. With custom dashboards, you can combine metrics from your Prometheus exporters into widgets, add PromQL-based queries, and organize the widgets to fit your monitoring needs. For more information, see Using CloudWatch dashboards.

Costs

We charge for Prometheus collectors by the hour, and CloudWatch OpenTelemetry metric ingestion pricing applies. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Amazon CloudWatch solution: Prometheus metric collection on Amazon ECS

This solution helps you collect Prometheus-compatible metrics from your Amazon ECS tasks without running or scaling your own collection infrastructure. Your tasks expose metrics through Prometheus exporters, and a Amazon CloudWatch managed Prometheus collector discovers the tasks in your VPC through DNS-based service discovery, scrapes their `/metrics` endpoints, and delivers the metrics to CloudWatch. Because Amazon ECS tasks are ephemeral and their IP addresses change, this solution uses AWS Cloud Map service discovery so that the collector always scrapes the current set of tasks. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions. For more information about managed collectors, see Amazon CloudWatch managed Prometheus collectors.

Topics

- Requirements
- Enable Prometheus exporters
- Step 1: Set up security groups
- Step 2: Set up the scrape configuration
- Step 3: Set up the scraper
- Validate metrics collection
- Build a custom dashboard
- Costs

Requirements

This solution is relevant for the following conditions:

- Compute: Amazon ECS tasks (on Amazon EC2 or AWS Fargate) that expose Prometheus-compatible metrics.
- Amazon ECS services that you register with AWS Cloud Map for DNS-based service discovery.
- Amazon VPC with DNS enabled, and at least two subnets in different Availability Zones for the collector.
- A security group that allows the collector to reach the metrics port on your tasks. If your tasks and the collector share a security group, add a self-referencing ingress rule for the metrics port.

Enable Prometheus exporters

A Prometheus exporter is a process that exposes metrics in Prometheus exposition format on an HTTP `/metrics` endpoint. On Amazon ECS, you typically run an exporter as a sidecar container in the same task definition as your application, or instrument the application to expose metrics directly. The following exporters are commonly used:

- **Node Exporter** — Host-level infrastructure metrics for tasks on the Amazon EC2 launch type (default port 9100). For more information, see the Node Exporter repository on GitHub.
- **JMX Exporter** — JVM and Java application metrics for Java workloads. For more information, see the JMX Exporter repository on GitHub.
- **NGINX Prometheus Exporter** — NGINX web server and reverse proxy metrics. For more information, see the NGINX Prometheus Exporter repository on GitHub.
- **DCGM Exporter** — NVIDIA GPU metrics for GPU workloads (default port 9400). For more information, see the DCGM Exporter repository on GitHub.
- **HAProxy Exporter** — HAProxy load balancer metrics. For more information, see the HAProxy Exporter repository on GitHub.
- **Apache Exporter** — Apache HTTP Server metrics. For more information, see the Apache Exporter repository on GitHub.

Add the exporter to your task definition and expose its metrics port by following the exporter's official documentation. For a complete catalog of available exporters, see Exporters and integrations in the Prometheus documentation.

Step 1: Set up security groups

Confirm that the security group for your Amazon ECS tasks allows inbound traffic from the collector on the metrics port. When your tasks and the collector share a security group, add a self-referencing ingress rule so that resources in the same security group can communicate. The following command allows inbound traffic on port 80 from the same security group:

```
aws ec2 authorize-security-group-ingress \
  --group-id sg-security-group-id \
  --ip-permissions IpProtocol=tcp,FromPort=80,ToPort=80,UserIdGroupPairs='[{GroupId=sg-security-group-id}]'
```

Step 2: Set up the scrape configuration

For Amazon ECS tasks that you register with AWS Cloud Map, use DNS-based service discovery (`dns_sd_configs`) to automatically find and scrape your tasks as they start and stop. The `dns_sd_configs` section directs the collector to query the AWS Cloud Map DNS name and scrape all returned IP addresses. When Amazon ECS replaces a task, the collector picks up the change on its next DNS query. The `relabel_configs` section adds consistent labels to your metrics so that you can query and filter across services. For the full set of supported configuration options, see Scraper configuration.

```
global:
  scrape_interval: 30s
  scrape_timeout: 10s

scrape_configs:
  - job_name: 'ecs-payforadoption'
    dns_sd_configs:
      - names: ['payforadoption-go.Workshop-space']
        type: A
        port: 80
    metrics_path: '/metrics'
    relabel_configs:
      - target_label: service_name
        replacement: 'payforadoption-go'
      - target_label: cloudmap_namespace
        replacement: 'Workshop-space'
      - target_label: environment
        replacement: 'production'
      - target_label: compute_platform
```

```
        replacement: 'ecs-fargate'
```

Step 3: Set up the scraper

Create a VPC-connected managed collector that scrapes your Amazon ECS tasks and delivers metrics to your CloudWatch dataset:

You can use GetDefaultScraperConfiguration to retrieve a general-purpose scraper configuration, or provide your own.

AWS API

Use the `CreateScraper` API operation to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
POST /scrapers HTTP/1.1

{
  "alias": "ecs-payforadoption-scraper",
  "source": {
    "vpcConfiguration": {
      "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
      "securityGroupIds": ["sg-security-group-id"]
    }
  },
  "destination": {
    "cloudWatchConfiguration": {
      "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
    }
  },
  "scrapeConfiguration": {
    "configurationBlob": "base64-encoded-blob"
  }
}
```

AWS CLI

Use the `create-scraper` command to create a scraper with a CloudWatch destination. Replace the subnet, security group, and dataset information with your own values.

```
aws amp create-scraper \
  --alias "ecs-payforadoption-scraper" \
```

```
--source '{
  "vpcConfiguration": {
    "subnetIds": ["subnet-subnet-id-1", "subnet-subnet-id-2"],
    "securityGroupIds": ["sg-security-group-id"]
  }
}' \
--scrape-configuration configurationBlob=$(cat ecs-scraper.yml | base64 -w 0) \
--destination '{
  "cloudWatchConfiguration": {
    "datasetArn": "arn:aws:cloudwatch:us-west-2:123456789012:dataset/default"
  }
}'
```

Validate metrics collection

After the collector begins delivering metrics, run an ad-hoc query in CloudWatch using Query Studio to confirm that your metrics arrive. For example, the following query returns the average requests per second to the Amazon ECS task over the last five minutes:

```
rate(payforadoption_requests_total[5m])
```

Build a custom dashboard

After the collector begins delivering your Amazon ECS metrics to CloudWatch, you can build a custom CloudWatch dashboard to visualize them. With custom dashboards, you can combine metrics from your Prometheus exporters into widgets, add PromQL-based queries, and organize the widgets to fit your monitoring needs. For more information, see Using CloudWatch dashboards.

Costs

We charge for Prometheus collectors by the hour, and CloudWatch OpenTelemetry metric ingestion pricing applies. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Amazon CloudWatch solution: Amazon MSK insights

This solution helps you monitor your Amazon MSK clusters in CloudWatch without running or scaling your own collection infrastructure. When you enable Open Monitoring on an Amazon MSK cluster, the cluster exposes Prometheus metrics through the JMX Exporter and Node Exporter,

and an Amazon CloudWatch managed Prometheus collector scrapes those metrics and delivers them to CloudWatch. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions. For the complete collector setup walkthrough, including scraper creation and the example scrape configuration, see Integrate Amazon MSK.

Topics

- Requirements
- Enable the managed collector
- Amazon MSK detailed monitoring levels
- View the automatic dashboard
- Costs

Requirements

This solution is relevant for the following conditions:

- An Amazon MSK cluster in provisioned mode, with standard or express brokers. Managed collectors do not support Amazon MSK Serverless clusters.
- Open Monitoring enabled on the cluster, which exposes the JMX Exporter (port 11001) and Node Exporter (port 11002).
- Amazon VPC with DNS enabled, and at least two subnets in different Availability Zones for the collector.
- A security group that allows the collector to reach broker ports 11001 and 11002.

Enable the managed collector

To collect Amazon MSK metrics with a managed collector, you enable Open Monitoring on your cluster and then create a VPC-connected collector that scrapes the broker endpoints and delivers metrics to your CloudWatch dataset. The collector connects to your VPC and scrapes all brokers using DNS-based service discovery, so your monitoring stays resilient to broker replacements and cluster scaling. For step-by-step instructions, see Integrate Amazon MSK.

> **Note**
>
> The `--open-monitoring` parameter alone exposes the Prometheus endpoints on ports 11001 and 11002. Enhanced monitoring tiers such as `PER_TOPIC_PER_PARTITION` are

separate from Open Monitoring and can incur additional charges, so set them only if you need that level of Amazon MSK metrics.

Amazon MSK detailed monitoring levels

Separately from the Prometheus metrics that the collector scrapes, Amazon MSK publishes its own metrics to CloudWatch at several monitoring levels. Each level builds on the previous one and provides more granular operational visibility. You choose a monitoring level when you create or update the cluster. The following list summarizes the levels. For the complete list of metrics at each level, see Amazon MSK metrics for monitoring with CloudWatch in the *Amazon MSK Developer Guide*.

- **Basic monitoring** (`DEFAULT`) — The default level, available at no additional cost. Includes cluster-level and broker-level metrics such as the number of brokers, number of partitions, bytes and messages in and out, CPU utilization, and network I/O.
- **Enhanced broker-level monitoring** (`PER_BROKER`) — Includes basic monitoring plus more granular per-broker metrics, at an additional cost.
- **Enhanced topic-level monitoring** (`PER_TOPIC_PER_BROKER`) — Includes enhanced broker-level monitoring plus per-topic metrics for a more granular view of topic performance across brokers, at an additional cost.
- **Enhanced partition-level monitoring** (`PER_TOPIC_PER_PARTITION`) — The most detailed level. Includes enhanced topic-level monitoring plus fine-grained per-partition metrics such as offset lag, at an additional cost.

Note

Amazon MSK publishes detailed monitoring metrics to classic CloudWatch metrics, which you can query with the `GetMetricData` API. To make detailed monitoring metrics available alongside the Open Monitoring Prometheus metrics, you need to enable OpenTelemetry enrichment. For more information, see OpenTelemetry metric enrichment.

View the automatic dashboard

After the collector begins delivering your Amazon MSK metrics to CloudWatch, the CloudWatch console provides an automatic dashboard named **MSK OTel**. To open it, see MSK OTel, or sign in to the AWS Management Console, open the CloudWatch console, choose **Dashboards** in the navigation pane, choose **Automatic dashboards**, and then choose **MSK OTel**. You can start monitoring your clusters without building any widgets or dashboards yourself. If the automatic dashboard meets your needs, you can use it as is. To build a tailored monitoring experience, you can add any of its widgets to a custom dashboard. For more information about custom dashboards, see Using CloudWatch dashboards.

Costs

We charge for Prometheus collectors by the hour, and CloudWatch OpenTelemetry metric ingestion pricing applies. Higher Amazon MSK monitoring levels also incur additional monthly costs. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Amazon CloudWatch solution: Amazon OpenSearch Service insights

This solution helps you monitor your Amazon OpenSearch Service domains in CloudWatch without running or scaling your own collection infrastructure. A Amazon CloudWatch managed Prometheus collector connects to your VPC, collects cluster, node, and index metrics from the domain, and delivers them to CloudWatch. For general information about all CloudWatch observability solutions, see CloudWatch observability solutions. For the complete collector setup walkthrough, including scraper creation and the example scrape configuration, see Integrate Amazon OpenSearch Service.

Topics

- Requirements
- Enable the managed collector
- Validate metrics collection
- Build a custom dashboard
- Costs

Requirements

This solution is relevant for the following conditions:

- An Amazon OpenSearch Service domain with VPC access. Managed collectors support only domains that have VPC access. Domains with public access are not supported.
- Amazon VPC with DNS enabled, and at least two subnets in different Availability Zones for the collector.
- A security group that allows the collector to reach your domain endpoint over HTTPS (port 443).

Enable the managed collector

To collect Amazon OpenSearch Service metrics with a managed collector, you create a VPC-connected collector that connects to your domain and delivers metrics to your CloudWatch dataset. You specify the domain to collect from in the `exporters` field of the request, and provide the networking in the `source` field. For step-by-step instructions, see Integrate Amazon OpenSearch Service.

Validate metrics collection

Within minutes, metrics begin flowing to your CloudWatch dataset. To confirm that your metrics arrive, run an ad-hoc query in CloudWatch using Query Studio. For example, the following query returns the cluster health status for your domain:

```
opensearch_cluster_health_status
```

Build a custom dashboard

After the collector begins delivering your Amazon OpenSearch Service metrics to CloudWatch, you can build a custom CloudWatch dashboard to visualize them. With custom dashboards, you can combine the collected metrics into widgets, add PromQL-based queries, and organize the widgets to fit your monitoring needs. For more information, see Using CloudWatch dashboards.

Costs

We charge for Prometheus collectors by the hour, and CloudWatch OpenTelemetry metric ingestion pricing applies. For more information about CloudWatch pricing, see Amazon CloudWatch Pricing.

Monitor across accounts and Regions

To enable unified monitoring across accounts, CloudWatch offers the following features:

- **CloudWatch cross-account observability**– facilitate observability within a single Region with the Observability Access Manager (OAM) service. You can link accounts and easily view metrics, logs, traces, and other telemetry between accounts. This helps you to unify observability in central monitoring accounts that view telemetry shared from source accounts, and operate on this shared telemetry as if it were native to the monitoring account.
- **Cross-account cross-Region CloudWatch console**– delivers a console experience that allows you to view dashboards, metrics, and alarms consoles of other accounts across Regions by toggling between accounts. After you set up the necessary permissions, you use an account selector integrated into the alarms, dashboards, and metrics consoles to view metrics, dashboards, and alarms in other accounts without having to log in and out of the accounts. By enabling this feature, you can also set up dashboards that contain cross-account cross-Region metrics for centralized visibility within an account.
- **Cross-account cross-Region centralization**– Amazon CloudWatch centralization capability streamlines log and metrics consolidation for improved centralized monitoring, analysis, and compliance across your entire AWS infrastructure. You define centralization rules that control what gets centralized, enabling unified monitoring, alarming, and analysis from a single destination. For more information, see Cross-account cross-Region log centralization and Cross-account cross-Region metrics centralization.

These features are complementary to each other and can be used independently or together. See the following table for a comparison of the features. We recommend that you use CloudWatch cross-account observability for the richest cross-account observability and discovery experience within a Region for your metrics, logs, and traces.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
What is it?	Unified access to underlying telemetry and other observability resources across	A designated monitoring account assume a **CrossAccountSharingRole** defined in source	Amazon CloudWatch centralization capability streamlines log and metrics consolida

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
	multiple accounts. After this is configured, observability resources are seamlessly viewable between accounts, eliminating the need for role assumptions. The central monitoring account gains direct access to the telemetry data and resources from source accounts, streamlining the monitoring and observability process.	accounts from the CloudWatch console. By assuming this role, the monitoring account can invoke operations such as dashboard viewing on behalf of source accounts, directly from its console.	tion for improved centralized monitoring, analysis, and compliance across your entire AWS infrastructure.

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
How does it work?	A monitoring account, using the Observability Access Monitoring service, creates a *sink* and attaches a sink policy to it. The sink policy defines which resources they would like to view and which source accounts should share them. Then source accounts can create a link to the monitoring account sink, establishing what they actually want to share. After the link is created, the specified resources are visible in the monitoring account.	A source account initiates the configuration by setting up a **CrossAccountSharingRole**, allowing a monitoring account to run operations in the source account. Then, a monitoring account enables the cross-account cross-Region selector in the console by specifying the source account ID. This enables the monitoring account to be able to switch into the source account. When switching, the CloudWatch console checks for the existence of a service-linked role that allows CloudWatch to assume the **CrossAccountSharingRole** that was created in the source account.	Uses AWS Organizations trusted access. An administrator creates a centralization rule in the management or delegated administrator account that defines source accounts/OUs/Regions and a destination account/Region. Logs and metrics arriving in source accounts are automatically copied to the destination with source metadata (account ID and Region) added.
What telemetry is supported?	• Metrics • Traces • Logs	• Metrics • Traces	• Logs • Metrics

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
What functionality is supported?	• Dashboarding • Alarms • Metrics Insights • Anomaly Detection • CloudWatch Logs insights • Application Insights • Other functionalities. For more details see CloudWatch cross-account observability.	• Console switching between accounts and Regions in metrics, alarms, and traces consoles. • Custom dashboards with metrics and alarms from other accounts and Regions For more details, see Cross-account cross-Region CloudWatch console.	Logs: CloudWatch Logs Insights, Subscription filters, Metric filters Metrics: Custom through PutMetric Data, Embedded Metric Format (EMF), OpenTelemetry (OTLP)
How many accounts can I use it with?	A monitoring account can see resources from as many as 100,000 source accounts at the same time. A source account can share their resources with as many as five different monitoring accounts.	By using the cross-account cross-Region selector in the console, a monitoring account can switch to one other account at a time but there is no limit on the number of accounts that can be linked. When defining cross-account dashboards and alarms, many source accounts can be referenced.	Works with AWS Organizations supported number of accounts

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
Does it move telemetry data?	No. Resources are shared between accounts with the exception of copied traces.	No. An IAM policy is configured to allow embedded account switching for cross-account cross-Region resource visibility.	Yes
How much does it cost?	No extra charges for shared logs and metrics, and the first trace copy is free. For more information about pricing, see Amazon CloudWatch pricing.	No additional charges for cross-account or cross-Region actions.	The first copy of centralized telemetry is free. An optional backup region (second copy) is a paid feature. For more information, see Amazon CloudWatch Pricing.
Does it support observability across Regions?	No	Yes	Yes, you can centralize data across Regions.
Does it support programmatic access?	Yes. the AWS CLI, AWS Cloud Development Kit (AWS CDK), and APIs are supported.	No	Yes
Does it support programmatic setup?	Yes	Yes	Yes

	CloudWatch cross-account observability	Cross-account cross-Region CloudWatch console	Cross-account cross-Region centralization
Does it support AWS Organizations?	Yes	Yes	Yes. AWS Organizations is required to use this feature.

Topics

- CloudWatch cross-account observability
- Cross-account cross-Region CloudWatch console
- Cross-account cross-Region metrics centralization

CloudWatch cross-account observability

With Amazon CloudWatch cross-account observability, you can monitor and troubleshoot applications that span multiple accounts within a Region. Seamlessly search, visualize, and analyze your metrics, logs, traces, Application Signals services and service level objectives (SLOs), Application Insights applications, and internet monitors in any of the linked accounts without account boundaries.

Set up one or more AWS accounts as *monitoring accounts* and link them with multiple *source accounts*. A monitoring account is a central AWS account that can view and interact with observability data generated from source accounts. A source account is an individual AWS account that generates observability data for the resources that reside in it. Source accounts share their observability data with the monitoring account. The shared observability data can include the following types of telemetry:

- Metrics in Amazon CloudWatch. You can choose to share the metrics from all namespaces with the monitoring account, or filter to a subset of namespaces.
- Log groups in Amazon CloudWatch Logs. You can choose to share all log groups with the monitoring account, or filter to a subset of log groups.
- Traces in AWS X-Ray
- Services and Service level objectives (SLOs) in Application Signals

- Applications in Amazon CloudWatch Application Insights
- Monitors in CloudWatch Internet Monitor

To create links between monitoring accounts and source accounts, you can use the CloudWatch console. Alternatively, use the *Observability Access Manager* commands in the AWS CLI and API. For more information, see Observability Access Manager API Reference.

A *sink* is a resource that represents an attachment point in a monitoring account. Source accounts can link to the sink to share observability data. Each account can have one sink per Region. Each sink is managed by the monitoring account where it is located. An *observability link* is a resource that represents the link established between a source account and a monitoring account. Links are managed by the source account.

The next topic explains how to set up CloudWatch cross-account observability in both monitoring accounts and source accounts. For information about the cross-account cross-Region CloudWatch dashboard, see Cross-account cross-Region CloudWatch console.

Use Organizations for source accounts

There are two options for linking source accounts to your monitoring account. You can use one or both options.

- Use AWS Organizations to link accounts in an organization or organizational unit to the monitoring account.
- Connect individual AWS accounts to the monitoring account.

We recommend that you use Organizations so that new AWS accounts created later in the organization are automatically onboarded to cross-account observability as source accounts.

Details about linking monitoring accounts and source accounts

- Each monitoring account can be linked to as many as 100,000 source accounts.
- Each source account can share data with as many as five monitoring accounts.
- You can set up a single account as both a monitoring account and a source account. If you do, this account sends only the observability data from itself to its linked monitoring account. It does not relay the data from its source accounts.
- A monitoring account specifies which telemetry types can be shared with it. A source account specifies which telemetry types it wants to share.

- If there are more telemetry types selected in the *monitoring account* than in the source account, the accounts are linked. Only the data types that are selected in both accounts are shared.
- If there are more telemetry types selected in the *source account* than in the monitoring account, the link creation fails and nothing is shared.
- A metric name doesn't appear in the monitoring account console until that metric emits new data points after the link is created.

- To remove a link between accounts, do so from the source account.
- To delete a sink in a monitoring account, you must first remove all links to that sink the monitoring account.

Pricing

Cross-account observability in CloudWatch comes with no extra cost for logs and metrics, Application Signals, and the first trace copy is free. For more information about pricing, see Amazon CloudWatch Pricing.

Contents

- Link monitoring accounts with source accounts
  - Necessary permissions
    - Permissions needed to create links
    - Permissions needed to monitor across accounts
  - Setup overview
  - Step 1: Set up a monitoring account
  - Step 2: (Optional) Download an CloudFormation template or URL
  - Step 3: Link the source accounts
    - Use an CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts
    - Use an CloudFormation template to set up individual source accounts
    - Use a URL to set up individual source accounts
- Manage monitoring accounts and source accounts
  - Link more source accounts to an existing monitoring account
  - Remove the link between a monitoring account and source account

- View information about a monitoring account

Link monitoring accounts with source accounts

The topics in this section explain how to set up links between monitoring accounts and source accounts.

We recommend that you create a new AWS account to serve as the monitoring account for your organization.

Contents

- Necessary permissions
  - Permissions needed to create links
  - Permissions needed to monitor across accounts
- Setup overview
- Step 1: Set up a monitoring account
- Step 2: (Optional) Download an CloudFormation template or URL
- Step 3: Link the source accounts
  - Use an CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts
  - Use an CloudFormation template to set up individual source accounts
  - Use a URL to set up individual source accounts

Necessary permissions

Permissions needed to create links

To create links between a monitoring account and a source account, you must be signed in with certain permissions.

- **To set up a monitoring account** – You must have either full administrator access in the monitoring account, or you must sign in to that account with the following permissions:

  JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowSinkModification",
            "Effect": "Allow",
            "Action": [
                "oam:CreateSink",
                "oam:DeleteSink",
                "oam:PutSinkPolicy",
                "oam:TagResource"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnly",
            "Effect": "Allow",
            "Action": ["oam:Get*", "oam:List*"],
            "Resource": "*"
        }
    ]
}
```

- **Source account, scoped to a specific monitoring account** – To create, update, and manage links for just one specified monitoring account, you must sign in to account with at least the following permissions. In this example, the monitoring account is 999999999999.

  If the link isn't going to share all seven resource types (metrics, logs, traces, Application Insights applications, Application Signals services and service level objectives (SLOs), and Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, `application-signals:Link`, or `internetmonitor:Link` as needed.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink",
                "oam:DeleteLink",
                "oam:GetLink",
```

```
                "oam:TagResource"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:link/*"
        },
        {
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Effect": "Allow",
            "Resource": "arn:*:oam:*:*:sink/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceAccount": [
                        "999999999999"
                    ]
                }
            }
        },
        {
            "Action": "oam:ListLinks",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
```

```
         },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
             "Action": "application-signals:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
    ]
}
```

- **Source account, with permissions to link to any monitoring account** – To create a link to any existing monitoring account sink and share metrics, log groups, traces, Application Insights applications, and Internet Monitor monitors, you must sign in to the source account with full administrator permissions or sign in there with the following permissions

  If the link isn't going to share all seven resource types (metrics, logs, traces, Application Insights applications, Application Signals services and service level objectives (SLOs), and Internet Monitor monitors), you can omit `cloudwatch:Link`, `logs:Link`, `xray:Link`, `applicationinsights:Link`, `application-signals:Link`, or `internetmonitor:Link` as needed.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "oam:CreateLink",
                "oam:UpdateLink"
            ],
            "Resource": [
                "arn:aws:oam:*:*:link/*",
                "arn:aws:oam:*:*:sink/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "oam:List*",
                "oam:Get*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:DeleteLink",
                "oam:GetLink",
                "oam:TagResource"
            ],
            "Resource": "arn:aws:oam:*:*:link/*"
        },
        {
            "Action": "cloudwatch:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "xray:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": "logs:Link",
            "Effect": "Allow",
            "Resource": "*"
        },
        {
             "Action": "applicationinsights:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
             "Action": "internetmonitor:Link",
             "Effect": "Allow",
             "Resource": "*"
        },
        {
             "Action": "application-signals:Link",
             "Effect": "Allow",
             "Resource": "*"
        }
```

```
    ]
}
```

Permissions needed to monitor across accounts

After a link has been created, to view source account information from a monitoring account, you must be signed in to an account with one of the following:

- Full administrator access in the monitoring account
- The following cross-account permissions, in addition to permissions to view the specific types of resources that you will be monitoring

```
{
   "Sid": "AllowReadOnly",
   "Effect": "Allow",
   "Action": [
     "oam:Get*",
     "oam:List*"
   ],
   "Resource": "*"
 }
```

Setup overview

The following high-level steps show you how to set up CloudWatch cross-account observability.

> **Note**
>
> We recommend creating a new AWS account to use as your organization's monitoring account.

1. Set up a dedicated monitoring account.
2. (Optional) Download an CloudFormation template or copy a URL to link source accounts.
3. Link source accounts to the monitoring account.

After completing these steps, you can use the monitoring account to view the observability data of the source accounts.

Step 1: Set up a monitoring account

Follow the steps in this section to set up an AWS account as a monitoring account for CloudWatch cross-account observability.

Prerequisites

- **If you're setting up accounts in an AWS Organizations organization as the source accounts** – Get the organization path or organization ID.
- **If you're not using Organizations for the source accounts** – Get the account IDs of the source accounts.

To set up an account as a monitoring account, you must have certain permissions. For more information, see Necessary permissions.

To set up a monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Configure**.
5. For **Select data**, choose whether this monitoring account will be able to view **Logs**, **Metrics**, **Traces**, **Application Insights - Applications**, **Internet Monitor - Monitors**, and **Application Signals - Services, Service Level Objectives (SLOs)** data from the source accounts it is linked to.
6. For **List source accounts**, enter the source accounts that this monitoring account will view. To identify the source accounts, enter individual account IDs, organization paths, or organization IDs. If you enter an organization path or organization ID, this monitoring account is allowed to view observability data from all linked accounts in that organization.

   Separate the entries in this list with commas.

> **Important**
>
> When you enter an organization path, follow the exact format. The ou-id must end with a / (a slash character). For example: `o-a1b2c3d4e5/r-f6g7h8i9j0example/ou-def0-awsbbbb/`

7. For **Define a label to use to identify your source account**, you can define alabel that is used to create a CloudFormation template. The label is then applied to source accounts when that template is used to link the source accounts to this monitoring account.

   You can specify whether to use account names or email addresses in this label, and also use variables such as `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain`.

   > **Note**
   >
   > In the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

8. Choose **Configure**.

> **Important**
>
> The link between the monitoring and source accounts is not complete until you configure the source accounts. For more information, see the following sections.

Step 2: (Optional) Download an CloudFormation template or URL

To link source accounts to a monitoring account, we recommend using an AWS CloudFormation template or a URL.

- **If you are linking an entire organization** – CloudWatch provides an CloudFormation template.
- **If you are linking individual accounts** – Use either an CloudFormation template or a URL that CloudWatch provides.

To use an CloudFormation template, you must download it during these steps. After you link the monitoring account with at least one source account, the CloudFormation template is no longer available to download.

To download an CloudFormation template or copy a URL for linking source accounts to the monitoring account

1. Sign in to the account that you want to use as a monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Resources to link accounts**.
5. Do one of the following:
   - Choose **AWS organization** to get a template to use to link accounts in an organization to this monitoring account.
   - Choose **Any account** to get a template or URL for setting up individual accounts as source accounts.
6. Do one of the following:
   - If you chose **AWS organization**, choose **Download CloudFormation template**.
   - If you chose **Any account**, choose either **Download CloudFormation template** or **Copy URL**.
7. (Optional) Repeat steps 5-6 to download both the CloudFormation template and the URL.

Step 3: Link the source accounts

Use the steps in these sections to link source accounts to a monitoring account.

To link monitoring accounts with source accounts, you must have certain permissions. For more information, see Necessary permissions.

Use an CloudFormation template to set up all accounts in an organization or an organizational unit as source accounts

These steps assume that you already downloaded the necessary CloudFormation template by performing the steps in Step 2: (Optional) Download an CloudFormation template or URL.

To use an CloudFormation template to link accounts in an organization or organizational unit to the monitoring account

1. Sign in to the organization's management account.
2. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. In the left navigation bar, choose **StackSets**.
4. Check that you are signed in to the Region that you want, then choose **Create StackSet**.
5. Choose **Next**.
6. Choose **Template is ready** and choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify StackSet details**, enter a name for the StackSet and choose **Next**.
10. For **Add stacks to stack set**, choose **Deploy new stacks**.
11. For **Deployment targets**, choose whether to deploy to the entire organization or to specified organizational units.
12. For **Specify regions**, choose which Regions to deploy CloudWatch cross-account observability to.
13. Choose **Next**.
14. On the **Review** page, confirm your selected options and choose **Submit**.
15. In the **Stack instances** tab, refresh the screen until you see that your stack instances have the status **CREATE_COMPLETE**.

Use an CloudFormation template to set up individual source accounts

These steps assume that you already downloaded the necessary CloudFormation template by performing the steps in Step 2: (Optional) Download an CloudFormation template or URL.

To use an CloudFormation template to set up individual source accounts for CloudWatch cross-account observability

1. Sign in to the source account.
2. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. In the left navigation bar, choose **Stacks**.

4. Check that you are signed in to the Region that you want, then choose **Create stack**, **With new resources (standard)**.
5. Choose **Next**.
6. Choose **Upload a template file**.
7. Choose **Choose file**, choose the template that you downloaded from the monitoring account, and choose **Open**.
8. Choose **Next**.
9. For **Specify stack details**, enter a name for the stack and choose **Next**.
10. On the **Configure stack options** page, choose **Next**.
11. On the **Review** page, choose **Submit**.
12. On the status page for your stack, refresh the screen until you see that your stack has the status **CREATE_COMPLETE**.
13. To use this same template to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-12.

Use a URL to set up individual source accounts

These steps assume that you already copied the necessary URL by performing the steps in Step 2: (Optional) Download an CloudFormation template or URL.

To use a URL to link individual source accounts to the monitoring account

1. Sign in to the account that you want to use as a source account.
2. Enter the URL that you copied from the monitoring account.

   You see the CloudWatch settings page, with some information filled in.
3. For **Select data**, choose whether this source account will share **Logs**, **Metrics**, **Traces**, **Application Insights - Applications**, and **Internet Monitor - Monitors** data to this monitoring account.

   For both **Logs** and **Metrics**, you can choose whether to share all resources or a subset with the monitoring account.

   a. (Optional) To share a subset of this account's log groups with the monitoring account, select **Logs** and choose **Filter Logs**. Then use the **Filter Logs** box to construct a query to

find the log groups that you want to share. The query will use the term `LogGroupName` and one or more of the following operands.

- = and `!=`
- `AND`
- `OR`
- `^` indicates LIKE and `!^` indicates NOT LIKE. These can be used only as prefix searches. Include a `%` at the end of the string that you want to search for and include.
- `IN` and `NOT IN`, using parentheses `(  )`

The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are `AND` and `OR`. There isn't a limit on the number of other operands.

> **Tip**
>
> Choose **View sample queries** to see the correct syntax for common query formats.

b. (Optional) To share a subset of this account's metric namespaces with the monitoring account, select **Metrics** and choose **Filter Metrics**. Then use the **Filter Metrics** box to construct a query to find the metric namespaces that you want to share. Use the term `Namespace` and one or more of the following operands.

- = and `!=`
- `AND`
- `OR`
- `LIKE` and `NOT LIKE`. These can be used only as prefix searches. Include a `%` at the end of the string that you want to search for and include.
- `IN` and `NOT IN`, using parentheses `(  )`

The complete query must be no more than 2000 characters and is limited to five conditional operands. Conditional operands are `AND` and `OR`. There isn't a limit on the number of other operands.

> **Tip**
>
> Choose **View sample queries** to see the correct syntax for common query formats.

4. Do not change the ARN in **Enter monitoring account configuration ARN**.
5. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account, if there is one. Optionally, choose **Edit** to change it.

> **Note**
>
> In the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AcccountEmail`, and `$AcccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

6. Choose **Link**.
7. Enter **`Confirm`** in the box and choose **Confirm**.
8. To use this same URL to link more source accounts to this monitoring account, sign out of this account and sign in to the next source account. Then repeat steps 2-7.

Manage monitoring accounts and source accounts

After you set up your monitoring accounts and source accounts, you can use the steps in these sections to manage them.

Contents

- Link more source accounts to an existing monitoring account
- Remove the link between a monitoring account and source account
- View information about a monitoring account

Link more source accounts to an existing monitoring account

Follow the steps in this section to add links from additional source accounts to an existing monitoring account.

Each source account can be linked to as many as five monitoring accounts. Each monitoring account can be linked to as many as 100,000 source accounts.

To manage a source account, you must have certain permissions. For more information, see Necessary permissions.

To add more source accounts to a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage source accounts**.
5. Choose the **Configuration policy** tab.
6. In the **Configuration policy** box, add the new source account ID in the **Principal** line.

   For example, suppose the **Principal** line is currently the following:

   ```
   "Principal": {"AWS": ["111111111111", "222222222222"]}
   ```

   To add 999999999999 as a third source account, edit the line to the following:

   ```
   "Principal": {"AWS": ["111111111111", "222222222222", "999999999999"]}
   ```
7. Choose **Update**.
8. Choose the **Configuration details** tab.
9. Choose the copy icon that is next to the monitoring account's sink ARN.
10. Sign in to the account that you want to use as a new source account.
11. Paste the monitoring account's sink ARN that you copied in Step 9.

    You see the CloudWatch settings page, with some information filled in.
12. For **Select data**, choose whether this source account will send **Logs**, **Metrics**, **Traces**, and **Application Insights - Applications**, **Internet Monitor - Monitors**, and **Application Signals - Services, Service Level Objectives (SLOs)** data to the monitoring accounts it is linked to.
13. Do not change the ARN in **Enter monitoring account configuration ARN**.
14. The **Define a label to identify your source account** section is pre-filled with the label choice from the monitoring account, if there is one. Optionally, choose **Edit** to change it.

> **Note**
>
> In the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, the only supported option is to use custom labels, and the `$AccountName`, `$AccountEmail`, and `$AccountEmailNoDomain` variables all resolve as *`account-id`* instead of the specified variable.

15. Choose **Link**.
16. Enter **`Confirm`** in the box and choose **Confirm**.

Remove the link between a monitoring account and source account

Follow the steps in this section to stop sending data from one source account to a monitoring account.

> **Note**
>
> After the source account stops sharing the metrics with the *Monitoring* account, the *Source* account metrics data is not accessible to the monitoring account. Source metric names can be visible to the monitoring account for upto 14 days.

You must have the permissions required to manage a source account to complete this task. For more information, see Necessary permissions.

To remove the link between a source account and a monitoring account

1. Sign in to the source account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Source account configuration**, choose **View linked monitoring accounts**.
5. Select the check box next to the monitoring account that you want to stop sharing data with.
6. Choose **Remove monitoring account**, **Confirm**.
7. Sign in to the monitoring account.
8. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

9. Choose **Settings**.
10. By **Monitoring account configuration**, choose **Manage monitoring account**.
11. In the **Configuration policy** box, delete the source account ID from the **Principal** line and choose **Update**.

View information about a monitoring account

Follow the steps in this section to view the cross-account settings for a monitoring account.

To manage a monitoring account, you must have certain permissions. For more information, see Necessary permissions.

To manage a monitoring account

1. Sign in to the monitoring account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the left navigation pane, choose **Settings**.
4. By **Monitoring account configuration**, choose **Manage monitoring accounts**.
5. To view the Observability Access Manager policy that enables this account to be a monitoring account, choose the **Configuration policy** tab.
6. To view the source accounts that are linked to this monitoring account, choose the **Linked source accounts** tab.
7. To view the monitoring account sink ARN, and the types of data that this monitoring account can view in linked source accounts, choose the **Linked source accounts** tab.

Cross-account cross-Region CloudWatch console

> **Note**
>
> We recommend that you use CloudWatch cross-account observability to get the richest cross-account observability and discovery experience for your metrics, logs, and traces within a Region. For more information, see CloudWatch cross-account observability.

The cross-account, cross-Region CloudWatch console allows you to easily switch between different accounts and Region by using selectors in the console to view the dashboards, alarms, and metrics

in other accounts and Regions. This feature also allows you to create cross-account, cross-Region dashboards which summarize your CloudWatch metrics from multiple AWS accounts and multiple Regions into a single dashboard, making them accessible without having to switch accounts or Regions.

Many organizations have their AWS resources deployed in multiple accounts, to provide billing and security boundaries. In this case, you should designate one or more of your accounts as your *monitoring accounts*, and build your cross-account cross-Region dashboards in these accounts. Cross-account cross-Region console functionality is integrated with AWS Organizations, to help you efficiently build your cross-account cross-Region dashboards.

The cross-account, cross-region CloudWatch console experience does not provide cross-account cross-Region visibility for logs. Additionally, it does not support the creation of alarms on metrics in other accounts or Regions from within a monitoring account.

Topics

- Enabling cross-account cross-Region functionality in CloudWatch
- (Optional) Integrate with AWS Organizations
- Troubleshooting your CloudWatch cross-account setup
- Monitoring account permissions for cross-account access
- Disabling and cleaning up after using cross-account

Enabling cross-account cross-Region functionality in CloudWatch

To set up cross-account cross-Region functionality in your CloudWatch console, use the CloudWatch console to set up your sharing accounts and monitoring accounts.

Set up a sharing account

You must enable sharing in each account that will make data available to the monitoring account.

This grants the read-only permissions that you choose in step 5 to all users that view a cross account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To enable your account to share CloudWatch data with other accounts

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. In the navigation pane, choose **Settings**.
3. For **Share your CloudWatch data**, choose **Configure**.
4. For **Sharing**, choose **Specific accounts** and enter the IDs of the accounts that you want to share data with.

   Any accounts that you specify here can view your account's CloudWatch data. Specify the IDs only of accounts that you know and trust.
5. For **Permissions**, specify how to share your data with one of the following options:

   - **Provide read-only access to your CloudWatch metrics, dashboards, and alarms**. This option enables the monitoring accounts to create cross-account dashboards that include widgets that contain CloudWatch data from your account.
   - **Include CloudWatch automatic dashboards**. If you select this option, users in the monitoring account can also view the information in this account's automatic dashboards. For more information, see Getting started with CloudWatch automatic dashboards.
   - **Include X-Ray read-only access for the X-Ray Trace Map**. If you select this option, users in the monitoring account can also view the X-Ray trace map and X-Ray trace information in this account. For more information, see Using the X-Ray Trace Map.
   - **Include read-only access for Database Insights**. If you select this option, users in the monitoring account can also view Database Insights telemetry in this account. For more information, see Set up cross-account cross-region monitoring for CloudWatch Database Insights.
   - **Full read-only access to everything in your account**. This option enables the accounts that you use for sharing to create cross-account dashboards that include widgets that contain CloudWatch data from your account. It also enables those accounts to look deeper into your account and view your account's data in the consoles of other AWS services.
6. Choose **Launch CloudFormation template**.

   In the confirmation screen, type **`Confirm`**, and choose **Launch template**.
7. Select the **I acknowledge...** check box, and choose **Create stack**.

Sharing with an entire organization

Completing the preceding procedure creates an IAM role which enables your account to share data with one account. You can create or edit an IAM role that shares your data with all accounts in an organization. Do this only if you know and trust all accounts in the organization.

This grants the read-only permissions listed in the policies shown in step 5 of the previous procedure to all users that view a cross-account dashboard in the account that you share with, if the user has corresponding permissions in the account that you share with.

To share your CloudWatch account data with all accounts in an organization

1. If you haven't already, complete the preceding procedure to share your data with one AWS account.
2. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Roles**.
4. In the list of roles, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.

   You see a policy like this:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

6. Change the policy to the following, replacing *org-id* with the ID of your organization.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "org-id"
        }
      }

    }
  ]
}
```

7. Choose **Update Trust Policy**.

Set up a monitoring account

Enable each monitoring account to view cross-account CloudWatch data. You can enable a monitoring account by using the Amazon CloudWatch console or the AWS Cloud Development Kit (AWS CDK).

After you set up a monitoring account, you can create cross-account dashboards. For more information, see Creating a customized CloudWatch dashboard.

Enable a monitoring account by using the console

When you complete the following procedure, CloudWatch creates a service role that CloudWatch uses in the monitoring account to access data shared from your other accounts. This service role is called **ServiceRoleForCloudWatchCrossAccountV2**.

To enable your account to view cross-account CloudWatch data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Settings**, and then, in the **Cross-account cross-region** section, choose **Configure**.
3. Under the **View cross-account cross-region** section, choose **Enable**, and then select the **Show selector in the console** checkbox to enable an account selector to appear in the CloudWatch console when you're graphing a metric or creating an alarm.
4. Under **View cross-account cross-region**, choose one of the following options:

- **Account Id Input**. This option prompts you to manually input an account ID each time that you want to switch accounts when you view cross-account data.
- **AWS Organization account selector**. This option causes the accounts that you specified when you completed your cross-account integration with Organizations to appear. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

  To do this, you must have first used your organization management account to allow CloudWatch to see a list of accounts in your organization. For more information, see (Optional) Integrate with AWS Organizations.
- **Custom account selector**. This option prompts you to enter a list of account IDs. When you next use the console, CloudWatch displays a dropdown list of these accounts for you to select from when you are viewing cross-account data.

  You can also enter a label for each of these accounts to help you identify them when choosing accounts to view.

  The account selector settings that a user makes here are retained only for that user, not for all other users in the monitoring account.

5. Choose **Enable**.

Enable a monitoring account by using the CDK

You can also set up a monitoring account without using the console. Define the **ServiceRoleForCloudWatchCrossAccountV2** role as infrastructure as code. Use this approach when you manage multiple monitoring accounts. It deploys the role consistently across all of them. The following example uses the AWS CDK in TypeScript to define the role with the required trust policy and an inline permissions policy:

```
import { Role, ServicePrincipal, PolicyDocument, PolicyStatement, Effect } from 'aws-
cdk-lib/aws-iam';

new Role(this, 'ServiceRoleForCloudWatchCrossAccountV2', {
  roleName: 'ServiceRoleForCloudWatchCrossAccountV2',
  path: '/service-role/',
  assumedBy: new ServicePrincipal('cloudwatch-crossaccount.amazonaws.com'),
  description: 'Allows CloudWatch to assume the CloudWatch-CrossAccountSharingRole in
 sharing accounts.',
```

```
  inlinePolicies: {
    CrossAccountAccess: new PolicyDocument({
      statements: [
        new PolicyStatement({
          effect: Effect.ALLOW,
          actions: ['sts:AssumeRole'],
          resources: ['arn:aws:iam::*:role/CloudWatch-CrossAccountSharingRole'],
        }),
      ],
    }),
  },
});
```

This example uses a wildcard resource, which is the least restrictive scope. To restrict the monitoring account to specific sharing accounts or to an organization, change the `resources` array. You can add a `conditions` property with an `aws:ResourceOrgID` condition. Either approach matches one of the inline policies in the following section.

To deploy the role to many accounts at once, use the synthesized template with CloudFormation StackSets. A stack set creates the role across multiple accounts and Regions in a single operation. A stack set also automatically deploys the role to new accounts that join your organization. For more information about CloudFormation StackSets, see Working with CloudFormation StackSets in the *CloudFormation User Guide*.

Scope of accounts the monitoring account can access

When you define the role with the AWS CDK, the inline permissions policy grants `sts:AssumeRole` permission. CloudWatch uses this permission to assume the **CloudWatch-CrossAccountSharingRole** in your sharing accounts. To follow the principle of least privilege, limit the `Resource` of the `sts:AssumeRole` action to only the accounts that the monitoring account needs. You can scope access in the following ways, from most restrictive to least restrictive:

- **An entire organization** – Allow the monitoring account to assume the sharing role only in accounts that belong to your organization. Replace *`org-id`* with the ID of your organization (for example, `o-a1b2c3d4e5`). Use the following JSON inline policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::*:role/CloudWatch-CrossAccountSharingRole",
      "Condition": {
        "StringEquals": {
          "aws:ResourceOrgID": "org-id"
        }
      }
    }
  ]
}
```

- **A list of accounts** – Allow the monitoring account to assume the sharing role only in the accounts that you list. Replace each example account ID (such as 111122223333) with the ID of a sharing account. Use the following JSON inline policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::111122223333:role/CloudWatch-CrossAccountSharingRole",
        "arn:aws:iam::444455556666:role/CloudWatch-CrossAccountSharingRole"
      ]
    }
  ]
}
```

- **Any account** – Allow the monitoring account to assume the sharing role in any account. This matches the permissions of the role that CloudWatch creates for you, but it is the least restrictive option. Use the following JSON inline policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::*:role/CloudWatch-CrossAccountSharingRole"
    }
  ]
```

```
}
```

Cross-Region functionality

This feature provides Cross-Region functionality automatically. You do not need to take any extra steps to display metrics from different Regions in a single account on the same graph or the same dashboard. Cross-Region functionality is not supported for alarms, so you cannot create an alarm in one Region that watches a metric in a different Region.

(Optional) Integrate with AWS Organizations

If you want to integrate cross-account functionality with AWS Organizations, you must make a list of all accounts in the organization available to the monitoring accounts.

To enable cross-account CloudWatch functionality to access a list of all accounts in your organization

1. Sign in to your organization's management account.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
3. In the navigation pane, choose **Settings**, then choose **Configure**.
4. For **Grant permission to view the list of accounts in the organization**, choose **Specific accounts** to be prompted to enter a list of account IDs. The list of accounts in your organization are shared with only the accounts that you specify here.
5. Choose **Share organization account list**.
6. Choose **Launch CloudFormation template**.

   In the confirmation screen, type **`Confirm`**, and choose **Launch template**.

Troubleshooting your CloudWatch cross-account setup

This section contains troubleshooting tips for cross-account, console deployment in CloudWatch.

I am getting access denied errors displaying cross-account data

Check the following:

- Your monitoring account must have a role named **ServiceRoleForCloudWatchCrossAccountV2**. If it does not, create this role. For more information, see the section called “Set up a monitoring account”.
- Each sharing account must have a role named **CloudWatch-CrossAccountSharingRole**. If it does not, create this role. For more information, see the section called “Set up a sharing account”.
- The sharing role must trust the monitoring account.

To confirm that your roles are set up properly for the CloudWatch cross-account console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, make sure the needed role exists. In a sharing account, look for **CloudWatch-CrossAccountSharingRole**. In a monitoring account, look for **ServiceRoleForCloudWatchCrossAccountV2**.
4. If you are in a sharing account and **CloudWatch-CrossAccountSharingRole** already exists, choose **CloudWatch-CrossAccountSharingRole**.
5. Choose **Trust relationships**, **Edit trust relationship**.
6. Confirm that the policy lists either the account ID of the monitoring account, or the organization ID of an organization that contains the monitoring account.

I don't see an account dropdown in the console

First, check that you have created the correct IAM roles, as discussed in the preceding troubleshooting section. If those are set up correctly, make sure that you have enabled this account to view cross-account data, as described in Enable Your Account to View Cross-Account Data.

Monitoring account permissions for cross-account access

To access an action in source accounts successfully, the user in the monitoring account must have the equivalent permission for all resources (*) for that action in the monitoring account. This is a local permission requirement in the monitoring account and is unrelated to permissions in the cross-account sharing role in source accounts.

Example

To start a Logs query in a source account, you must have wildcard (*) access to StartQuery in the monitoring account. The source account's cross-account role can still restrict access to specific log groups.

Supported - wildcard resource:

```
{
  "Effect": "Allow",
  "Action": "logs:StartQuery",
  "Resource": "*"
}
```

Not supported - specific ARN:

```
{
  "Effect": "Allow",
  "Action": "logs:StartQuery",
  "Resource": "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/my-function:*"
}
```

Disabling and cleaning up after using cross-account

To disable cross-account functionality for CloudWatch, follow these steps.

Step 1: Remove the cross-account stacks or roles

The best method is to remove the CloudFormation stacks that were used to enable cross-account functionality.

- In each of the sharing accounts, remove the **CloudWatch-CrossAccountSharingRole** stack.
- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, remove the **CloudWatch-CrossAccountListAccountsRole** stack in the organization's management account.

If you didn't use the CloudFormation stacks to enable cross-account functionality, do the following:

- In each of the sharing accounts, delete the **CloudWatch-CrossAccountSharingRole** IAM role.

- If you used AWS Organizations to enable cross-account functionality with all accounts in an organization, delete the **CloudWatch-CrossAccountSharing-ListAccountsRole** IAM role in the organization's management account.

Step 2: Remove the service role

In the monitoring account, delete the **ServiceRoleForCloudWatchCrossAccountV2** IAM service role.

Cross-account cross-Region metrics centralization

Amazon CloudWatch Metrics centralization automatically centralizes metrics from multiple source accounts and Regions into a single destination account within your AWS Organizations organization. You define rules to control what gets centralized, enabling unified monitoring, alarming, and analysis across your entire AWS infrastructure.

CloudWatch Metrics centralization supports the full range of CloudWatch metrics query capabilities in the destination account, including GetMetricData, PromQL, Metric Math, Anomaly Detection, and Alarms.

Metrics centralization concepts

Before you begin using CloudWatch Metrics centralization, familiarize yourself with the following concepts:

- **Source account** – The AWS account where metric data originates.
- **Destination account** – The AWS account where centralized metric data is stored. This account serves as the centralized location for metric querying, alarming, and analysis.
- **Source metadata** – Centralized metrics are automatically tagged with source metadata so you can identify the origin of each metric in the destination account. For Metrics Insights querying, the dimensions `:@aws.account` and `:@aws.region` are added. For PromQL querying, the attributes `@aws.account` and `@aws.region` are added.
- **Backup region** – An optional secondary Region (see pricing) within the destination account where metric data can be centralized for increased resiliency and disaster recovery purposes.

Centralized metrics

When you enable metrics centralization, CloudWatch automatically centralizes metrics from your source accounts to the destination account. Once centralized, a copy of the metrics is now owned by your destination account. Metrics ingested through PutMetricData or EMF are queryable using GetMetricData. Metrics ingested through OpenTelemetry (OTLP) are queryable using PromQL.

Supported metric types

The following metric types are centralized:

- Custom metrics (PutMetricData)
- Embedded Metric Format (EMF) metrics
- OpenTelemetry (OTLP) metrics

Source metadata dimensions

Centralized metrics include additional source identification metadata. The format differs based on the query path:

Metrics Insights querying (GetMetricData)

Dimension	Description
`:@aws.account`	The AWS account ID where the metric originated
`:@aws.region`	The AWS Region where the metric originated

PromQL querying

Attribute	Description
`@aws.account`	The AWS account ID where the metric originated
`@aws.region`	The AWS Region where the metric originated

Setting up metrics centralization

To set up CloudWatch Metrics centralization, you need to configure centralization rules that define how metric data flows from source accounts to your destination account.

Prerequisites

- AWS Organizations must be set up and the source and destination accounts must both belong to the same organization.
- Trusted access must be enabled for CloudWatch so the management account and the destination account can access the metric data.

> **Note**
>
> It is recommended to enable trusted access through the console, which automatically creates the required service-linked role (SLR). If trusted access is enabled through other methods, the service-linked role will need to be created separately.

Creating a centralization rule

Use the following procedure to create a centralization rule that centralizes metric data from source accounts to your destination account.

To create a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Configure rule**.
5. Specify source details by setting the following fields, then choose **Next**:
    a. **Centralization rule name**: Enter a unique name for the centralization rule.
    b. **Source accounts**: Define source selection criteria to pick accounts from which metric data will be centralized. The selection criteria can include:

- A list of member accounts in the organization
- A list of organization units in the organization
- The entire organization

You can provide the selection criteria in two modes:

- **Builder**: A click-based experience to generate the source selection criteria
- **Editor**: A free-form text box to provide the source selection criteria

Supported syntax for source selection criteria:

- **Supported Keys:**`OrganizationId` | `OrganizationalUnitId` | `AccountId` | `*`
- **Supported Operators:**= | `IN` | `OR`

c. **Source regions**: Select a list of Regions to look for the metric data to centralize.

6. Specify destination details by setting the following fields, then choose **Next**:

a. **Metrics**: Make sure Metrics is selected (it is enabled by default). If you only want to centralize metrics, you can deselect Logs.

b. **Destination region**: Select a primary Region that stores a copy of the centralized metric data.

7. Specify telemetry data by setting the following fields, then choose **Next**:

a. **Metrics**: All metrics from the source accounts are centralized to the destination account. This includes custom metrics, Embedded Metric Format (EMF) metrics, and OpenTelemetry (OTLP) metrics.

b. **Backup region**: Optionally select a Region that stores a second copy of the centralized metric data. Logs and metrics can have separate backup region settings.

> **Note**
>
> Currently, all metrics from source accounts are centralized. Selective metric filtering is not supported at this time.

8. Review the centralization rule, optionally make any last-minute edits, and choose **Create centralization policy**.

Modifying a centralization rule

To modify a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. Select the rule to update and choose **Edit**.
6. Update the rule configuration as needed, choosing **Next** to proceed through each step.
7. In Step 4, Review and configure, choose **Update centralization policy**.

Viewing a centralization rule

To view a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.
3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. View a list of all existing centralization rules and choose a specific rule name to view its details.

Deleting a centralization rule

To delete a centralization rule

1. Navigate to the CloudWatch console in the Management or Delegated Administrator account of the organization.
2. Choose **Settings**.

3. Navigate to the **Organization** tab.
4. Choose **Manage rules**.
5. Select the rule to delete and choose **Delete**.
6. Confirm deletion and choose **Delete**.

Features supported with centralized metrics

The following CloudWatch features work with centralized metrics in the destination account:

> **Note**
>
> Resource-based automatic dashboards (such as EC2 and S3) have partial support. These dashboards may show incomplete data for centralized metrics because they depend on resource metadata that is not centralized from source accounts.

Feature	Description
GetMetricData API	Query metric data points programmatically
GetMetricStatistics API	Query metric statistics
ListMetrics API	Discover available centralized metrics
Console Metric Browser	Browse and navigate centralized metrics
Metrics Insights (SQL queries)	Query metrics using SQL-like syntax
Query Studio	Unified query interface for PromQL and metrics
Search Expressions	Dynamic metric discovery through SEARCH()
Metric Math	Arithmetic, comparison, and logical operators on time series
Anomaly Detection	ML-based anomaly detection models and alarms
Metric Alarms	Standard threshold-based alarms
Composite Alarms	Boolean logic combining multiple alarm states

Feature	Description
PromQL Alarms	Alarms using PromQL expressions
Alarm Actions (SNS)	SNS notifications on alarm state changes
CloudWatch Dashboards	Add centralized metrics to dashboards
Metric Streams	Stream centralized metrics to Firehose, S3, or partners
PromQL Querying	Prometheus-compatible metric queries

Monitoring and troubleshooting centralization rules

You can monitor the status and performance of your centralization rules using CloudWatch metrics, the CloudWatch console, and AWS CloudTrail logs.

Monitoring centralization API calls with AWS CloudTrail

AWS CloudTrail logs API calls made to the centralization service. Key CloudTrail events include:

- `CreateCentralizationRuleForOrganization`: When a new centralization rule is created
- `UpdateCentralizationRuleForOrganization`: When an existing rule is modified
- `DeleteCentralizationRuleForOrganization`: When a rule is deleted
- `GetCentralizationRuleForOrganization`: When rule details are retrieved
- `ListCentralizationRulesForOrganization`: When rules are listed

Troubleshooting common issues

If metrics are not being centralized as expected, review the following common scenarios:

- **Historical metric data** – The centralization feature only processes new metric data that arrives after rule creation. Historical data is not centralized.
- **Trusted access not enabled** – Trusted access must be enabled for CloudWatch in AWS Organizations for the management account and the destination account.
- **Source selection criteria** – Verify that your centralization rule's source selection criteria includes the correct accounts and Regions.

- **Organization membership** – Both source and destination accounts must belong to the same AWS Organizations organization.
- **Metric quota limits** – If the destination account has reached its metric quota limits, new metrics cannot be ingested. Request a quota increase if needed.
- **Rule health status** – Check the centralization rule health status in the console or using the `GetCentralizationRuleForOrganization` API. Each centralization rule has a health status that indicates whether it is operating correctly. Rule health statuses include:
  - `HEALTHY`: The rule is operating normally and replicating metric data as configured.
  - `UNHEALTHY`: The rule has encountered issues and may not be replicating data correctly.
  - `PROVISIONING`: Centralization for the organization is in the process of being set up.

Pricing

The first copy of centralized metrics is free. Please see the Amazon CloudWatch Pricing page for more information.

Related resources

- Configure CloudWatch integration with AWS Organizations
- CloudWatch cross-account observability
- Creating metrics from log events using filters

Analyzing, optimizing, and reducing CloudWatch costs

This section describes how Amazon CloudWatch features generate costs. It also provides methods that can help you analyze, optimize, and reduce CloudWatch costs. Throughout this section, this guide sometimes refers to pricing when describing CloudWatch features. For information about pricing, see Amazon CloudWatch pricing.

Topics

- Analyze CloudWatch cost and usage data with Cost Explorer
- Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena
- Optimizing and reducing costs of CloudWatch metrics
- Optimizing and reducing costs of CloudWatch alarms
- Optimizing and reducing costs of CloudWatch Container Insights
- Optimizing and reducing costs of CloudWatch Database Insights
- Optimizing and reducing costs of CloudWatch Logs

Analyze CloudWatch cost and usage data with Cost Explorer

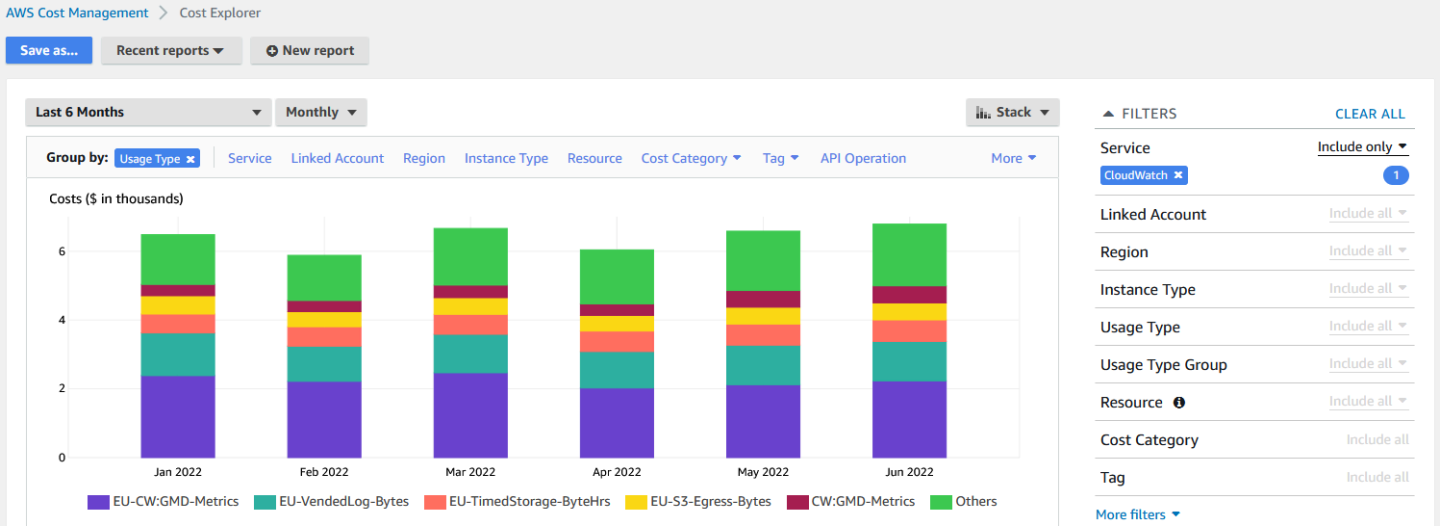
With AWS Cost Explorer, you can visualize and analyze cost and usage data for AWS services over time, including CloudWatch. For more information, see Getting started with AWS Cost Explorer.

The following procedure describes how to use Cost Explorer to visualize and analyze CloudWatch cost and usage data.

To visualize and analyze CloudWatch cost and usage data

1. Sign in to the Cost Explorer console at https://console.aws.amazon.com/cost-management/home#/custom.
2. Under **FILTERS**, for **Service**, select **CloudWatch**.
3. For **Group by**, choose **Usage Type**. You can also group your results by other categories, such as the following:
   - **API Operation** – See which API operations generated the most costs.
   - **Region** – See which Regions generated the most costs.

The following image shows an example of the costs that CloudWatch features generated over six months.



To see which CloudWatch features generated the most costs, look at the values for `UsageType`. For example, `EU-CW:GMD-Metrics` represents the costs that CloudWatch bulk API requests generated.

> **Note**
>
> The strings for `UsageType` match specific features and Regions. For example, the first part of `EU-CW:GMD-Metrics` (EU) matches the Europe (Ireland) Region, and the second part of `EU-CW:GMD-Metrics` (`GMD-Metrics`) matches CloudWatch bulk API requests.
>
> The entire string for `UsageType` can be formatted as follows: `<Region>-CW:<Feature>` or `<Region>-<Feature>`.
>
> Some CloudWatch features such as logs and alarms also uses the `Global` Region to identify the Free Tier usage. For example, `Global-DataScanned-Bytes` represents free CloudWatch Logs data ingestion usage.
>
> To enhance readability, the strings for `UsageType` in the tables throughout this document have been shortened to their string suffixes. For example, `EU-CW:GMD-Metrics` is shortened to `GMD-Metrics`.

The following table includes the names of each CloudWatch feature, lists the names of each sub-feature, and lists the strings for `UsageType`.

CloudWatch feature	***CloudWatch sub-feature***	**UsageType**
CloudWatch metrics	*Custom metrics*	`MetricMonitorUsage`
	Detailed monitoring	`MetricMonitorUsage`
	Embedded metrics	`MetricMonitorUsage`
	OpenTelemetry metrics (observations)	`OTEL:Values`
	OpenTelemetry metrics (ingestion)	`OTEL:Bytes`
		`OTEL:Bytes:EKS-CI`
		`OTEL:Bytes:AWS`
	PromQL queries	`PromQL:SamplesScanned`
	PromQL alarms	`PromQLAlarmMonitorUsage`
	Metrics centralization	`CentralizedBytes`
	Metrics centralization (free tier)	`CentralizedBytesFree`
CloudWatch API requests	*API requests*	`Requests`
	Bulk (Get)	`GMD-Metrics`

CloudWatch feature	***CloudWatch sub-feature***	**`UsageType`**
	Contributor Insights	`GIRR-Metrics`
	Bitmap image snapshot	`GMWI-Metrics`
CloudWatch metric streams	*Metric streams*	`MetricStreamUsage`
CloudWatch dashboards	*Dashboard with 50 or fewer metrics*	`DashboardsUsageHour-Basic`
	Dashboard with more than 50 metrics	`DashboardsUsageHour`
CloudWatch alarms	*Standard (metric alarm)*	`AlarmMonitorUsage`
	High resolution (metric alarm)	`HighResAlarmMonitorUsage`
	Metrics Insights query alarm	`MetricInsightAlarmUsage`
	Composite (aggregated alarm)	`CompositeAlarmMonitorUsage`
Container Insights	Enhanced observability for Amazon EKS	`ObservationUsage`
	Enhanced observability for Amazon ECS	`MetricsUsage`

CloudWatch feature	***CloudWatch sub-feature***	**`UsageType`**
CloudWatch Application Signals	*Application Signals with Transaction Search*	`Application-Signals-Bytes, XRay-Spans-Indexed`
	Application Signals with X-Ray	`Application-Signals`
CloudWatch custom logs	*Collect (data ingestion for Standard log class)*	`DataProcessing-Bytes`
	Collect (data ingestion for Infrequent Access log class)	`DataProcessingIA-Bytes`
	Analyze (query)	`DataScanned-Bytes`
	Analyze (Live Tail)	`Logs-LiveTail`
	Store (Standard)	`TimedStorage-ByteHrs`
	Store (Infrequent Access)	`TimedStorage-IA-ByteHrs`
	Store (Archive Instant Access)	`TimedStorage-AIA-ByteHrs`
	Detect and mask (data protection)	`DataProtection-Bytes`
CloudWatch vended logs	*Delivery (Amazon CloudWatch Logs Standard log class)*	`VendedLog-Bytes`

CloudWatch feature	***CloudWatch sub-feature***	**`UsageType`**
	Delivery (CloudWatch Logs Infrequent Access log class)	`VendedLogIA-Bytes`
	Delivery (Amazon S3)	`S3-Egress-Bytes`
	Delivery (Amazon S3) in Parquet format	`S3-Egress-InputBytes`
	Delivery (Amazon Data Firehose)	`FH-Egress-Bytes`
Contributor Insights	*CloudWatch Logs (Rules)*	`ContributorInsightRules`
	CloudWatch Logs (Events)	`ContributorInsightEvents`
	Amazon DynamoDB (Rules)	`ContributorRulesManaged`
	DynamoDB Events)	`ContributorEventsManaged`
Database Insights	*Serverless*	`DatabaseInsights-ACU-Hours`
	Provisioned	`DatabaseInsights-vCPU-Hours`
	Limitless	`DatabaseInsights-ACU-Hours`
Canaries (Synthetics)	*Run*	`Canary-runs`

CloudWatch feature	***CloudWatch sub-feature***	**`UsageType`**
RUM	*Events*	`RUM-event`
Network monitoring	*Network Synthetic Monitor*	`CWNMHybrid-Paid`
	Internet Monitor (monitored resources)	`InternetMonitor-MonitoredResource`
	Internet Monitor (monitored city networks)	`InternetMonitor-CityNetwork`

Analyze CloudWatch cost and usage data with AWS Cost and Usage Reports and Athena

Another way to analyze CloudWatch cost and usage data is by using AWS Cost and Usage Reports with Amazon Athena. AWS Cost and Usage Reports contain a comprehensive set of cost and usage data. You can create reports that track your costs and usage, and you can publish these reports to an S3 bucket of your choice. You also can download and delete your reports from your S3 bucket. For more information, see What are AWS Cost and Usage Reports? in the *AWS Cost and Usage Reports User Guide*.

> **Note**
>
> There is no charge for using AWS Cost and Usage Reports. You only pay for storage when you publish your reports to Amazon Simple Storage Service (Amazon S3). For more information, see Quotas and restrictions in the *AWS Cost and Usage Reports User Guide*.

Athena is a query service that you can use with AWS Cost and Usage Reports to analyze cost and usage data. You can query your reports in your S3 bucket without needing to download them first. For more information, see What is Amazon Athena? in the *Amazon Athena User Guide*. For more

information, see What is Amazon Athena? in the *Amazon Athena User Guide*. For information about pricing, see Amazon Athena pricing.

The following procedure describes the process for enabling AWS Cost and Usage Reports and integrating the service with Athena. The procedure contains two example queries that you can use to analyze CloudWatch cost and usage data.

> **Note**
>
> You can use any of the example queries in this document. All of the example queries in this document correspond to a database named ***costandusagereport***, and show results for the month of April and the year 2025. You can change this information. However, before you run a query, make sure that the name of your database matches the name of the database in the query.

To analyze cost and usage data with AWS Cost and Usage Reports and Athena

1. Enable AWS Cost and Usage Reports. For more information, see Creating cost and usage reports in the *AWS Cost and Usage Reports User Guide*.

   > **Tip**
   >
   > When you create your reports, make sure to select **Include resource IDs**. Otherwise, your reports won't include the column `line_item_resource_id`. This line helps you further identify costs when analyzing cost and usage data.

2. Integrate AWS Cost and Usage Reports with Athena. For more information, see Setting up Athena using CloudFormation templates in the *AWS Cost and Usage Reports User Guide*.
3. Query your cost and usage reports.

Example of Athena query to show CloudWatch costs per month

You can use the following query to show which CloudWatch features generated the most costs for a given month.

```
SELECT
```

```
CASE
-- Metrics
WHEN line_item_usage_type LIKE '%%MetricMonitorUsage%%' THEN 'Metrics (Custom, Detailed
 monitoring management portal EMF)'
WHEN line_item_usage_type LIKE '%%Requests%%' THEN 'Metrics (API Requests)'
WHEN line_item_usage_type LIKE '%%GMD-Metrics%%' THEN 'Metrics (Bulk API Requests)'
WHEN line_item_usage_type LIKE '%%MetricStreamUsage%%' THEN 'Metric Streams'
-- Contributor Insights
WHEN line_item_usage_type LIKE '%%Contributor%%' THEN 'Contributor Insights'
-- Dashboard
WHEN line_item_usage_type LIKE '%%DashboardsUsageHour%%' THEN 'Dashboards'
-- Alarms
WHEN line_item_usage_type LIKE '%%AlarmMonitorUsage%%' THEN 'Alarms (Standard)'
WHEN line_item_usage_type LIKE '%%HighResAlarmMonitorUsage%%' THEN 'Alarms (High
 Resolution)'
WHEN line_item_usage_type LIKE '%%MetricInsightAlarmUsage%%' THEN 'Alarms (Metrics
 Insights)'
WHEN line_item_usage_type LIKE '%%CompositeAlarmMonitorUsage%%' THEN 'Alarms
 (Composite)'
-- Container Insights with enhanced observability
WHEN (line_item_usage_type LIKE '%%MetricsUsage%%' OR line_item_usage_type LIKE '%
%ObservationUsage%%') THEN 'Container Insights (Enhanced Observability)'
-- OpenTelemetry metrics
WHEN line_item_usage_type LIKE '%%CW:OTEL:Bytes%' THEN 'OpenTelemetry Metrics'
-- PromQL
WHEN line_item_usage_type LIKE '%%CW:PromQL:SamplesScanned%%' THEN 'PromQL Queries'
WHEN line_item_usage_type LIKE '%%CW:PromQLAlarmMonitorUsage%%' THEN 'PromQL Alarms'
-- Metrics Centralization
WHEN line_item_usage_type LIKE '%%CW:CentralizedBytes%%' THEN 'Metrics Centralization'
-- Database Insights
WHEN line_item_usage_type LIKE '%%DatabaseInsights%%' THEN 'Database Insights'
-- Logs
WHEN line_item_usage_type LIKE '%%DataProcessing-Bytes%%' THEN 'Logs (Collect - Data
 Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProcessingIA-Bytes%%' THEN 'Infrequent Access
 Logs (Collect - Data Ingestion)'
WHEN line_item_usage_type LIKE '%%DataProtection-Bytes%%' THEN 'Logs (Data Protection -
 Detect and Mask)'
WHEN line_item_usage_type LIKE '%%TimedStorage-ByteHrs%%' THEN 'Logs (Storage -
 Standard)'
WHEN line_item_usage_type LIKE '%%TimedStorage-IA-ByteHrs%%' THEN 'Logs (Storage -
 Infrequent Access)'
WHEN line_item_usage_type LIKE '%%TimedStorage-AIA-ByteHrs%%' THEN 'Logs (Storage -
 Archive Instant Access)'
```

```
WHEN line_item_usage_type LIKE '%%DataScanned-Bytes%%' THEN 'Logs (Analyze - Logs
 Insights queries)'
WHEN line_item_usage_type LIKE '%%Logs-LiveTail%%' THEN 'Logs (Analyze - Logs Live
 Tail)'
-- Vended Logs
WHEN line_item_usage_type LIKE '%%VendedLog-Bytes%%' THEN 'Vended Logs (Delivered to
 CW)'
WHEN line_item_usage_type LIKE '%%VendedLogIA-Bytes%%' THEN 'Vended Infrequent Access
 Logs (Delivered to CW)'
WHEN line_item_usage_type LIKE '%%FH-Egress-Bytes%%' THEN 'Vended Logs (Delivered to
 Data Firehose)'
WHEN (line_item_usage_type LIKE '%%S3-Egress%%') THEN 'Vended Logs (Delivered to S3)'
-- Network Monitoring
WHEN line_item_usage_type LIKE '%%CWNMHybrid-Paid%%' THEN 'Network Monitor'
WHEN line_item_usage_type LIKE '%%InternetMonitor%%' THEN 'Internet Monitor'
-- Other
WHEN line_item_usage_type LIKE '%%Application-Signals%%' THEN 'Application Signals'
WHEN line_item_usage_type LIKE '%%Canary-runs%%' THEN 'Synthetics'
WHEN line_item_usage_type LIKE '%%RUM-event%%' THEN 'RUM'
ELSE 'Others'
END AS UsageType,
-- REGEXP_EXTRACT(line_item_resource_id,'^(?:.+?:){5}(.+)$',1) as ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
 SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can remove this comment at the beginning of the line and specify
 an AWS account.
GROUP BY
1
ORDER BY TotalSpend DESC,
UsageType;
```

Example of Athena query to show how CloudWatch features generated costs

You can use the following query to show the results for `UsageType` and `Operation`. This shows you how CloudWatch features generated costs. The results also show the values for `UsageQuantity` and `TotalSpend`, so that you can see your total usage costs.

> **Tip**
>
> For more information about `UsageType`, add the following line to this query:
> `line_item_line_item_description`
> This line creates a column called ***Description***.

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation
```

Optimizing and reducing costs of CloudWatch metrics

Many AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2) , Amazon S3, and Amazon Data Firehose, automatically send metrics to CloudWatch at no charge. However, metrics that are grouped in the following categories can incur additional costs:

- ***Custom metrics, detailed monitoring, and embedded metrics***
- ***API requests***
- ***Metric streams***

For more information, see Using Amazon CloudWatch metrics.

Custom metrics

You can create custom metrics to organize data points in any order and at any rate.

All custom metrics are prorated by the hour. They're metered only when they're sent to CloudWatch. For information about how metrics are priced, see Amazon CloudWatch Pricing.

The following table lists the names of relevant sub-features for CloudWatch metrics. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify metric-related costs.

> **Note**
>
> To get more details about the metrics that are listed in the following table while you're querying cost and usage data with Athena, match the strings for `Operation` with the results that are shown for `line_item_operation`.

The following table lists CloudWatch metrics sub-features with their UsageType and Operation strings.

CloudWatch sub-feature	**`UsageType`**	**`Operation`**	**Purpose**
Custom metrics			Custom metrics

CloudWatch sub-feature	`UsageType`	`Operation`	Purpose
	`MetricMonitorUsage`	`MetricStorage`	
Detailed monitoring	`MetricMonitorUsage`	`MetricStorage:AWS/{Service}`	Detailed monitoring
Embedded metrics	`MetricMonitorUsage`	`MetricStorage:AWS/Logs-EMF`	Logs embedded metrics
Log filters	`MetricMonitorUsage`	`MetricStorage:AWS/CloudWatchLogs`	Log group metric filters

Detailed monitoring

CloudWatch has two types of monitoring:

- ***Basic monitoring***

  Basic monitoring is free and automatically enabled for all AWS services that support the feature.
- ***Detailed monitoring***

  Detailed monitoring incurs costs and adds different enhancements depending on the AWS service. For each AWS service that supports detailed monitoring, you can choose whether to enable it for that service. For more information, see Basic and detailed monitoring.

> **Note**
>
> Other AWS services support detailed monitoring and might refer to this feature using a different name. For example, detailed monitoring for Amazon S3 is referred to as ***request metrics***.

Similar to custom metrics, detailed monitoring is prorated by the hour and metered only when data is sent to CloudWatch. Detailed monitoring generates costs by the number of metrics that are sent to CloudWatch. To reduce costs, only enable detailed monitoring when necessary. For information about how detailed monitoring is priced, see Amazon CloudWatch Pricing.

Example: Athena query

You can use the following query to show which EC2 instances have detailed monitoring enabled.

```
SELECT
bill_payer_account_id as Payer,
line_item_usage_account_id as LinkedAccount,
line_item_usage_type AS UsageType,
line_item_operation AS Operation,
line_item_resource_id AS ResourceID,
SUM(CAST(line_item_usage_amount AS double)) AS UsageQuantity,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='4'
AND line_item_operation='MetricStorage:AWS/EC2'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
bill_payer_account_id,
line_item_usage_account_id,
line_item_usage_type,
line_item_resource_id,
line_item_operation,
line_item_line_item_description
ORDER BY line_item_operation
```

Embedded metrics

With the CloudWatch embedded metric format, you can ingest application data as log data, so that you can generate actionable metrics. For more information, see Ingesting high-cardinality logs and generating metrics with the CloudWatch embedded metric format.

Embedded metrics generate costs by the number of logs ingested, number of logs archived, and number of custom metrics generated.

The following table lists the names of relevant sub-features for the CloudWatch embedded metric format. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify costs.

CloudWatch subfeature	**`UsageType`**	**`Operation`**	**Purpose**
Custom metrics	`MetricMonitorUsage`	`MetricStorage:AWS/Logs-EMF`	Logs embedded metrics
Logs ingestion	`DataProcessing-Bytes`	`PutLogEvents`	Uploads a batch of log events to the specified log group or log stream
Logs archival (Standard)	`TimedStorage-ByteHrs`	`HourlyStorageMetering`	Stores logs per hour and logs per byte in CloudWatch Logs
Logs archival (Infrequent Access)	`TimedStorage-IA-ByteHrs`	`HourlyStorageMetering`	Stores Infrequent Access tier logs per hour and per byte in CloudWatch Logs
Logs archival (Archive Instant Access)	`TimedStorage-AIA-ByteHrs`	`HourlyStorageMetering`	Stores Archive Instant Access tier logs per hour and per byte in CloudWatch Logs

To analyze costs, use AWS Cost and Usage Reports with Athena so that you can identify which metrics are generating costs and determine how the costs are generated.

To make the most of costs generated by the CloudWatch embedded metric format, avoid creating metrics based on high-cardinality dimensions. This way, CloudWatch doesn't create a custom metric for each unique dimension combination. For more information, see Dimensions.

API requests

CloudWatch has the following types of API requests:

- ***API requests***
- ***Bulk (Get)***
- ***Contributor Insights***
- ***Bitmap image snapshot***

API requests generate costs by the request type and number of metrics requested.

The following table lists the types of API requests and includes the strings for `UsageType` and `Operation`, which can help you analyze and identify API-related costs.

API request type	**`UsageType`**	**`Operation`**	**Purpose**
API requests	`Requests`	`GetMetricStatistics`	Retrieves statistics for the specified metrics
	`Requests`	`ListMetrics`	Lists the specified metrics
	`Requests`	`PutMetricData`	Publishes metric data points to CloudWatch
	`Requests`	`GetDashboard`	Displays details for the specified dashboards
	`Requests`	`ListDashboards`	Lists the dashboards in your account
	`Requests`	`PutDashboard`	Creates or updates a dashboard

API request type	UsageType	Operation	Purpose
	Requests	DeleteDashboards	Deletes all specified dashboards
Bulk (Get)	GMD-Metrics	GetMetricData	Retrieves CloudWatch metric values
Contributor Insights	GIRR-Metrics	GetInsightRuleReport	Returns time-series data that's collected by a Contributor Insights rule
Bitmap image snapshot	GMWI-Metrics	GetMetricWidgetImage	Retrieves a snapshot of one or more CloudWatch metrics as a bitmap image

To analyze costs, use Cost Explorer, and group your results by ***API Operation***.

The billing console shows generic API requests under UsageType *Requests*. These appear as *X.XX USD per 1,000 requests - [region]*. This rate applies to all requests with the UsageType Requests, aggregated together, beyond your free tier allowance.

Costs for API requests vary, and you incur costs when you exceed the number of API calls provided to you under the AWS Free Tier limit.

> **Note**
>
> Only API requests with UsageType *Requests* are included in the AWS Free Tier limit. API requests with any other UsageType incur charges from the first call. For more information, see Using the AWS Free Tier in the *AWS Billing User Guide*.

The API requests that typically drive cost are `Put` and `Get` requests.

To monitor API request origins and identify users within your account, enable data events in CloudTrail and analyze recorded events using either:

- Amazon CloudWatch Logs with Log Insights
- Amazon S3 with Amazon Athena

> **Note**
>
> Data events are not logged automatically by trails and event data stores. Logging data events incurs additional costs. For more information, see AWS CloudTrail Pricing.

For more information, see Logging data events and Identifying resources driving CloudWatch GetMetricData charges using AWS CloudTrail.

API calls not incurring charges

When you log CloudWatch data events in CloudTrail, you might see more calls than you initiated. This happens because logging CloudWatch data events in CloudTrail captures API actions from internal components. Internal component calls don't incur CloudWatch charges. However, these events count toward your CloudTrail event logging total and may affect CloudTrail charges.

For example, CloudTrail will record GetMetricData calls initiated by a monitoring account to retrieve data from a source account, as well as GetMetricData calls initiated by CloudWatch dashboards to refresh widget data. These API calls do not incur CloudWatch charges.

PutMetricData

Each CloudWatch `PutMetricData` API call incurs charges. Frequent calls can significantly increase your costs, especially in high-volume monitoring scenarios. To reduce costs, consider batching multiple metrics in each API call or adjusting your monitoring frequency. For more information, see PutMetricData in the *Amazon CloudWatch API Reference*.

To make the most of costs that are generated by `PutMetricData`, batch more data into your API calls. Depending on your use case, consider using CloudWatch Logs or the CloudWatch embedded metric format to inject metric data. For more information, see the following resources:

- What is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*
- Ingesting high-cardinality logs and generating metrics with CloudWatch embedded metric format
- Lowering costs and focusing on our customers with Amazon CloudWatch embedded custom metrics

GetMetricData

The CloudWatch `GetMetricData` API operation can also significantly increase your costs. Third-party monitoring tools often increase costs when they frequently pull data to generate insights. To learn more about pricing and best practices for using `GetMetricData`, see GetMetricData in the *Amazon CloudWatch API Reference*.

To reduce costs generated by `GetMetricData`, consider only pulling data that's monitored and used, or consider pulling data less often. Depending on your use case, you might consider using metric streams instead of `GetMetricData`, so that you can push data in near real time to third parties at a lower cost. For more information, see the following resources:

- Using metric streams
- CloudWatch Metric Streams - Send AWS Metrics to Partners and to Your Apps in Real Time

GetMetricStatistics

Depending on your use case, you might consider using `GetMetricStatistics` instead of `GetMetricData`. With `GetMetricData`, you can retrieve data quickly and at scale. However, `GetMetricStatistics` is included under the AWS Free Tier limit for up to one million API requests, which can help you reduce costs if you don't need to retrieve as many metrics and data points per call. For more information, see the following resources:

- GetMetricStatistics in the *Amazon CloudWatch API Reference*
- Should I use GetMetricData or GetMetricStatistics?

> **Note**
>
> External callers make API calls. For APIs that are supported by CloudTrail data events (such as **GetMetricData** and **GetMetricWidgetImage**), you can use CloudTrail to identify the top CloudWatch API callers and potentially mitigate or identify unexpected calls. For more information, see How to use CloudTrail to analyse your CloudWatch API Usage. For other CloudWatch APIs not supported by CloudTrail, you can open a technical support request to the CloudWatch team and ask for information about them. For information about creating a technical support request, see How do I get technical support from AWS?.

CloudWatch metric streams

With CloudWatch metric streams, you can send metrics continuously to AWS destinations and third-party service provider destinations.

Metric streams generate costs by the number of metric updates. Metric updates always include values for the following statistics:

- `Minimum`
- `Maximum`
- `Sample Count`
- `Sum`

For more information, see Statistics that can be streamed.

To analyze costs that are generated by CloudWatch metric streams, use AWS Cost and Usage Reports with Athena. This way, you can identify which metric streams are generating costs and determine how the costs are generated.

Example: Athena query

You can use the following query to track which metric streams generate costs by Amazon Resource Name (ARN).

```
SELECT
SPLIT_PART(line_item_resource_id,'/',2) AS "Stream Name",
line_item_resource_id as ARN,
SUM(CAST(line_item_unblended_cost AS decimal(16,2))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='4'
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
-- AND line_item_usage_account_id = '123456789012' - If you want to filter on a
 specific account, you can
remove this comment at the beginning of the line and specify an AWS account.
```

```
AND line_item_usage_type LIKE '%%MetricStreamUsage%%'
GROUP BY line_item_resource_id
ORDER BY TotalSpend DESC
```

To reduce costs generated by CloudWatch metric streams, stream only the metrics that bring your business value. You also can stop or pause any metric stream that you're not using.

Optimizing and reducing costs of CloudWatch alarms

With CloudWatch alarms, you can create alarms based on a single metric, alarms based on a Metrics Insights query, and composite alarms which watch other alarms.

> **Note**
>
> Costs for metric and composite alarms are prorated by the hour. You incur costs for your alarms only while your alarms exist. To optimize cost, be sure not to leave behind misconfigured or low-value alarms. To help with this, you can automate the cleanup of CloudWatch alarms that you no longer need. For more information, see Automating Amazon CloudWatch Alarm Cleanup at Scale

Metric alarms

Metric alarms have the following resolution settings:

- **Standard** (evaluated every 60 seconds)
- **High resolution** (evaluated every 10 seconds)

When you create a metric alarm, your costs are based on your alarm's resolution setting and the number of metrics that your alarm references. For example, a metric alarm that references one metric incurs one alarm-metric cost per hour. For more information, see Using Amazon CloudWatch alarms.

If you create a metric alarm that contains a metric math expression, which references multiple metrics, you incur a cost for each alarm-metric that's referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see Creating a CloudWatch alarm based on a metric math expression.

If you create an anomaly detection alarm, where your alarm analyzes past metric data to create a model of expected values, you incur a cost for each alarm-metric that's referenced in your alarm plus two additional metrics, one for the upper and lower band metrics that the anomaly detection model creates. For information about how to create an anomaly detection alarm, see Creating a CloudWatch alarm based on anomaly detection.

Metrics Insights query alarms

Metric Insights query alarms are a specific type of metric alarm, only available with standard resolution (evaluated every 60 seconds).

When you create a Metric Insights query alarm, your costs are based on the number of metrics analyzed by the query that your alarm references. For example, a Metric Insights query alarm that references a query whose filter matches ten metrics incurs ten metrics analyzed cost per hour. For more information, see the pricing example on Amazon CloudWatch Pricing.

If you create an alarm that contains both a Metrics Insights query and a metric math expression, it is reported as a Metrics Insights query alarm. If your alarm contains a metric math expression which references other metrics in addition to the metrics analyzed by the Metrics Insights query, you incur an additional cost for each alarm-metric that's referenced in the metric math expression. For information about how to create a metric alarm that contains a metric math expression, see Creating a CloudWatch alarm based on a metric math expression.

Composite alarms

Composite alarms contain rule expressions that specify how they should evaluate the states of other alarms to determine their own states. Composite alarms incur a standard cost per hour, regardless of how many other alarms they evaluate. Alarms that composite alarms reference in rule expressions incur separate costs. For more information, see Creating a composite alarm.

Alarm usage types

The following table lists the names of relevant sub-features for CloudWatch alarms. The table includes the strings for `UsageType`, which can help you analyze and identify alarm-related costs.

CloudWatch sub-feature	**`UsageType`**
Standard metric alarm	`AlarmMonitorUsage`

CloudWatch sub-feature	**UsageType**
High-resolution metric alarm	`HighResAlarmMonitorUsage`
Metrics Insights query alarm	`MetricInsightAlarmUsage`
Composite alarm	`CompositeAlarmMonitorUsage`

Reducing alarm costs

To optimize costs generated by metric math alarms that aggregate four or more metrics, you can aggregate data before the data is sent to CloudWatch. This way, you can create an alarm for a single metric instead of an alarm that aggregates data for multiple metrics. For more information, see Publishing custom metrics.

To optimize costs generated by Metrics Insights query alarms, you can make sure that the filter used for the query matches only the metrics that you want to monitor.

The best way to reduce costs is to remove all unnecessary or unused alarms. For example, you can delete alarms that evaluate metrics emitted by AWS resources that no longer exist.

Example of using DescribeAlarms to check for alarms in INSUFFICIENT_DATA state

If you delete a resource, but not the metric alarms that the resource emits, the alarms still exist and typically will go into the `INSUFFICIENT_DATA` state. To check for alarms that are in the `INSUFFICIENT_DATA` state, use the following AWS Command Line Interface (AWS CLI) command.

```
aws cloudwatch describe-alarms –state-value INSUFFICIENT_DATA
```

For more information, see Automating Amazon CloudWatch Alarm Cleanup at Scale.

Other ways to reduce costs include the following:

- Make sure to create alarms for the correct metrics.
- Make sure that you don't have any alarms enabled in Regions where you're not working.
- Remember that, although composite alarms reduce noise, they also generate additional costs.
- When deciding whether to create a standard alarm or high-resolution alarm, consider your use case and the value that each type of alarm brings.

Optimizing and reducing costs of CloudWatch Container Insights

CloudWatch Container Insights offers both standard and enhanced observability features for monitoring containerized applications in Amazon ECS and Amazon EKS. CloudWatch Container Insights leverages the embedded metric format to ingest telemetry from your container environments.

Container Insights with standard observability:

Standard Container Insights collects and visualizes metrics aggregated at the cluster and node levels. You can get started with standard mode of Container Insights using either the CloudWatch agent or the AWS Distro for Open Telemetry (ADOT). Using ADOT enables you to customize which metrics and dimensions are sent to CloudWatch.

Metrics in Container Insights are treated as "embedded metrics". Costs associated with these metrics are reflected in the usage types `MetricStorage:AWS/Logs-EMF` and `DataProcessing-Bytes`. For detailed pricing information, see the Embedded Metrics section at Amazon CloudWatch pricing.

Container Insights with enhanced observability:

Detailed visibility comes with Container Insights with enhanced observability, which delivers granular telemetry up to the pod and container levels in your applications. Similar to standard Container Insights, enhanced observability also comes with a standard set of critical metrics where you can get started by using the CloudWatch Observability add-on running on the CloudWatch agent. Container Insights offers enhanced observability with a new observation-based pricing in order to ensure cost effective bills justifying the benefit. See Amazon CloudWatch pricing for more information.

Here are the following UsageType and Operation associated with this Container Insights with enhanced observability:

CloudWatch sub-feature	**UsageType**	**Operation**
Container Insights with enhanced observability for Amazon EKS	`ObservationUsage`	`ObservationCount:CI-EKS`

CloudWatch sub-feature	UsageType	Operation
Container Insights with enhanced observability for Amazon ECS	MetricsUsage	MetricStorage:CI-ECS

Optimizing and reducing costs of CloudWatch Database Insights

CloudWatch Database Insights offers both standard and enhanced observability features for monitoring Amazon Aurora databases at both the instance and fleet level;. CloudWatch Database Insights consolidates logs and metrics from your applications, your databases, and the operating systems on which they run into a unified view in the console.

Database Insights Standard mode: Standard Database Insights is part of the AWS Free Tier and provides a rolling 7 days of performance data history for the database load metric.

Database Insights Advanced mode: Advanced Database Insights consolidates database metrics, SQL query analytics, and logs for Amazon Aurora and RDS databases into a unified experience in CloudWatch. Pricing is based on the amount of compute resources used by the monitored databases.

For details about how Database Insights is priced and pricing examples, see Amazon CloudWatch pricing.

Here are the UsageTypes and Operations associated with Database Insights:

UsageType	Operation	Instance configuration type	Database engine type
DatabaseInsights-vCPU-Hours	Aurora-MySQL:Provisioned	Provisioned	Aurora-MySQL
DatabaseInsights-ACU-Hours	Aurora-MySQL:Serverless	Serverless	Aurora-MySQL

UsageType	Operation	Instance configuration type	Database engine type
DatabaseInsights-vCPU-Hours	Aurora-PostgreSQL: Provisioned	Provisioned	Aurora-PostgreSQL
DatabaseInsights-ACU-Hours	Aurora-PostgreSQL: Serverless	Serverless	Aurora-PostgreSQL
DatabaseInsights-ACU-Hours	Aurora-PostgreSQL: Limitless	Limitless	Aurora-PostgreSQL

Optimizing and reducing costs of CloudWatch Logs

Amazon CloudWatch Logs has the following log types:

- ***Custom logs*** *(logs that you create for your applications)*
- ***Vended logs*** *(logs that other AWS services, such as Amazon Virtual Private Cloud (Amazon VPC) and Amazon Route 53, create on your behalf)*

For more information about vended logs, see Enabling logging from certain AWS services in the *Amazon CloudWatch Logs User Guide*.

Custom and vended logs generate costs based on the number of logs that are *collected*, *stored*, and *analyzed*. Separately, vended logs generate costs for delivery to Amazon S3 and Firehose.

The following table lists the names of the CloudWatch Logs features and names of relevant sub-features. The table includes the strings for `UsageType` and `Operation`, which can help you analyze and identify log-related costs.

CloudWatch Logs feature	*CloudWatch Logs sub-feature*	UsageType	Operation	Purpose
Custom logs	*Collect (data ingestion for Standard log class)*	DataProcessing-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream in a Standard class log group.
	Collect (data ingestion for Infrequent Access log class)	DataProcessingIA-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream in an Infrequent Access class log group.
	Detect and mask (data protection)	DataProtection-Bytes	PutLogEvents	Detects and masks protected data in log events.
	Store (Standard)	TimedStorage-ByteHrs	HourlyStorageMetering	Stores Standard tier logs per hour and per byte in CloudWatch Logs.
	Store (Infrequent Access)	TimedStorage-IA-ByteHrs	HourlyStorageMetering	Stores Infrequent Access tier logs per hour and per byte in CloudWatch Logs.

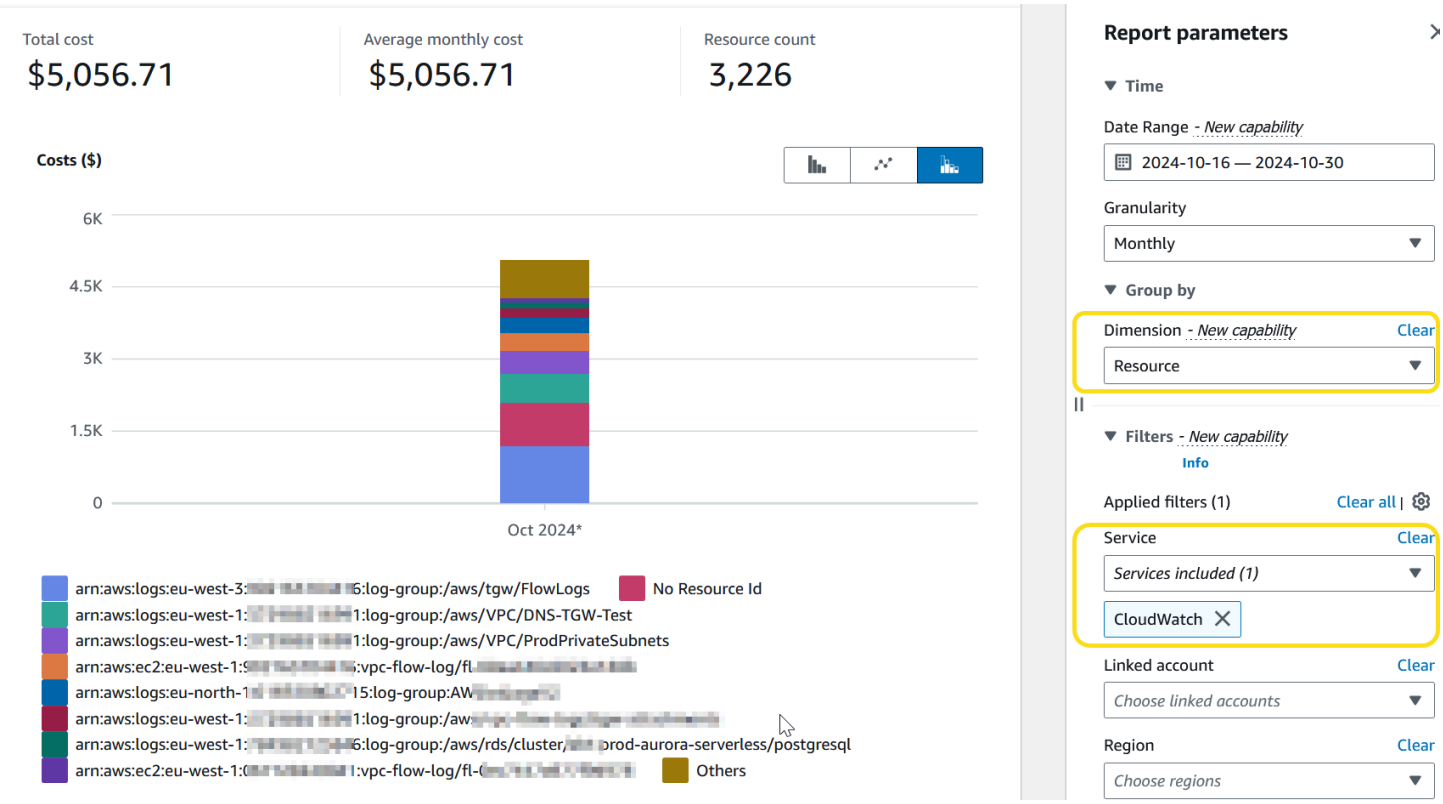
CloudWatch Logs feature	*CloudWatch Logs sub-feature*	UsageType	Operation	Purpose
	Store (Archive Instant Access)	TimedStorage-AIA-ByteHrs	HourlyStorageMetering	Stores Archive Instant Access tier logs per hour and per byte in CloudWatch Logs.
	Analyze (Logs Insights queries)	DataScanned-Bytes	StartQuery	Logs data scanned by CloudWatch Logs Insights queries
	Analyze (Logs Live Tail)	Logs-LiveTail	StartLiveTail	Logs analyzed during a CloudWatch Logs Live Tail session
Vended logs	*Delivery (CloudWatch Logs Standard log class)*	VendedLog-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream in a log group in the Standard log class.

CloudWatch Logs feature	*CloudWatch Logs sub-feature*	UsageType	Operation	Purpose
	Delivery (CloudWatch Logs Infrequent Access log class)	VendedLog IA-Bytes	PutLogEvents	Uploads a batch of logs to a specific log stream in a log group in the Infrequent Access log class.
	Delivery (Amazon S3)	S3-Egress-Bytes	LogDelivery	Uploads a batch of vended logs to a specific S3 bucket
	Delivery (Amazon S3) in Parquet format	S3-Egress-InputBytes	ParquetConversion	Perform Parquet conversion on logs delivered to Amazon S3
	Delivery (Firehose)	FH-Egress-Bytes	LogDelivery	Uploads a batch of vended logs to Amazon Data Firehose

To analyze costs, use AWS Cost Explorer Service or AWS Cost and Usage Reports with Athena. With either method, you can identify which logs are generating costs and determine how the costs are generated.

Using AWS Cost Explorer Service

Select **CloudWatch** for the **Service** filter and select **Resource** for the **Dimension**. When you select **Resource** as the dimension in Cost Explorer Service you can see only the past 14 days of usage.



Using Amazon Athena query to track logs that generate costs

You can use the following query to track which logs generate costs by resource ID.

```
SELECT
line_item_resource_id AS ResourceID,
line_item_usage_type AS Operation,
SUM(CAST(line_item_unblended_cost AS decimal(16,8))) AS TotalSpend
FROM
costandusagereport
WHERE
product_product_name = 'AmazonCloudWatch'
AND year='2025'
AND month='4'
AND line_item_operation IN
 ('PutLogEvents','HourlyStorageMetering','StartQuery','LogDelivery','StartLiveTail','ParquetCon
AND line_item_line_item_type NOT IN
 ('Tax','Credit','Refund','EdpDiscount','Fee','RIFee')
GROUP BY
line_item_resource_id,
line_item_usage_type
ORDER BY
```

```
TotalSpend DESC
```

To make the most of costs that are generated by CloudWatch Logs, consider the following:

- Identify top log groups by spend per operation by using the previous query.
- Log only the events that bring your business value and choose an efficient log syntax. A verbose log syntax drives volume and thus cost. This helps you generate lower costs for ingestion.
- Change your log retention settings, so that you generate lower costs for storage. For more information, see Change log data retention in CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*.
- Consider using the Infrequent Access log class where appropriate. Infrequent Access logs offers fewer features than Standard class. Determine whether you need the additional features of Standard log class and understand the difference between both classes. For more information, see the blog article New CloudWatch Logs log class for infrequent access logs at a reduced price. While the Infrequent Access class supports fewer features, it is suitable for a majority of use cases.
- Run queries that CloudWatch Logs Insights automatically saves in your history. This way, you generate fewer costs for analysis. For more information, see View running queries or query history in the *Amazon CloudWatch Logs User Guide*.
- Use the CloudWatch agent to collect system and application logs and send them to CloudWatch. This way, you can collect only the log events that meet your criteria. For more information, see Amazon CloudWatch Agent adds Support for Log Filter Expressions.

To reduce costs for vended logs, consider your use case, and then determine whether your logs should be sent to CloudWatch or Amazon S3. For more information, see Logs sent to Amazon S3 in the *Amazon CloudWatch Logs User Guide*.

Tip

If you want to use metric filters, subscription filters, CloudWatch Logs Insights, and Contributor Insights, send vended logs to CloudWatch.
Alternatively, if you're working with VPC Flow Logs and using them for auditing and compliance purposes, send vended logs to Amazon S3.

For information about how to track charges that are generated by publishing VPC Flow Logs to S3 buckets, see Using AWS Cost and Usage Reports and cost allocation tags to understand VPC FLow Logs data ingestion in Amazon S3.

For additional information about how to make the most of costs that are generated by CloudWatch Logs, see Which log group is causing a sudden increase in my CloudWatch Logs bill?.

AWS usage metrics

CloudWatch collects metrics that track the usage of some AWS resources and APIs. These metrics are published in the `AWS/Usage` namespace. Usage metrics in CloudWatch allow you to proactively manage usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch anomaly detection, and configuring alarms that alert you when usage approaches a threshold.

Some AWS services integrate these usage metrics with Service Quotas. For these services, you can use CloudWatch to manage your account's use of your service quotas. For more information, see Visualizing your service quotas and setting alarms.

Topics

- Visualizing your service quotas and setting alarms
- AWS API usage metrics
- CloudWatch usage metrics

Visualizing your service quotas and setting alarms

For some AWS services, you can use the usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information about service quotas, see What Is Service Quotas in the *Service Quotas User Guide*.

If you are signed in to an account that is set up as a monitoring account in CloudWatch cross-account observability, you can use that monitoring account to visualize service quotas and set alarms for metrics in the source accounts that are linked to that monitoring account. For more information, see CloudWatch cross-account observability.

The following services integrate their usage metrics with Service Quotas:

- AWS CloudHSM
- Amazon Chime SDK
- Amazon CloudWatch

- Amazon CloudWatch Logs
- Amazon DynamoDB
- Amazon EC2
- Amazon Elastic Container Registry
- Elastic Load Balancing
- AWS Fargate
- AWS Fault Injection Service
- AWS Interactive Video Service
- AWS Key Management Service
- Amazon Data Firehose
- Amazon Location Service
- Amazon Managed Blockchain (AMB) Query
- Amazon SageMaker AI
- Multi-party approval

To visualize a service quota and optionally set an alarm

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **Browse** tab, choose **Usage**, and then choose **By AWS Resource**.

   The list of service quota usage metrics appears.
4. Select the check box next to one of the metrics.

   The graph displays your current usage of that AWS resource.
5. To add your service quota to the graph, do the following:

   a. Choose the **Graphed metrics** tab.

   b. Choose **Math expression**, **Start with an empty expression**. In the new row, under **Details**, enter **`SERVICE_QUOTA(m1)`**.

      A new line is added to the graph, displaying the service quota for the resource represented in the metric.

6. To see your current usage as a percentage of the quota, add a new expression or change the current **SERVICE_QUOTA** expression. The new expression to use is **`m1/SERVICE_QUOTA(m1)*100`**.
7. (Optional) To set an alarm that notifies you if you approach the service quota, do the following:

   a. On the row with **`m1/SERVICE_QUOTA(m1)*100`**, under **Actions**, choose the alarm icon. It looks like a bell.

      The alarm creation page appears.

   b. Under **Conditions**, make sure that **Threshold type** is **Static** and **Whenever Expression1 is** is set to **Greater**. Under **than**, enter **`80`**. This creates an alarm that goes into ALARM state when your usage exceeds 80 percent of the quota.

   c. Choose **Next**.

   d. On the next page, select an Amazon SNS topic or create a new one, and then choose **Next**. The topic you select is notified when the alarm goes to ALARM state.

   e. On the next page, enter a name and description for the alarm, and then choose **Next**.

   f. Choose **Create alarm**.

AWS API usage metrics

Most APIs that support AWS CloudTrail logging also report usage metrics to CloudWatch. API usage metrics in CloudWatch allow you to proactively manage API usage by visualizing metrics in the CloudWatch console, creating custom dashboards, detecting changes in activity with CloudWatch anomaly detection, and configuring alarms that alert when usage approaches a threshold.

You can use the following table to learn about services that report API usage metrics to CloudWatch. The table lists the values to use for the `Service` dimension, so you can see the usage metrics from that service. You can use the procedure in this section to view the list of a service's APIs that report usage metrics to CloudWatch.

Service	Value for the `Service` dimension
AWS Identity and Access Management Access Analyzer	`Access Analyzer`

Service	Value for the Service dimension
AWS Account Management	Account Management
Alexa for Business	A4B
Amazon API Gateway	API Gateway
AWS App Mesh	App Mesh
AWS AppConfig	AWS AppConfig
Amazon AppFlow	AppFlow
Application Auto Scaling	Application Auto Scaling
Application Discovery Service	Application Discovery Service
Amazon AppStream	AppStream
WorkSpaces Applications Image Builder	Image Builder
Amazon Athena	Athena
AWS Audit Manager	Audit Manager
AWS Backup	Backup
AWS Batch	Batch
Amazon Braket	Braket
AWS Budgets	Budgets
AWS Certificate Manager	Certificate Manager
Amazon Chime SDK	ChimeSDK
Amazon Cloud Directory	Cloud Directory
AWS Cloud Map	Cloud Map

Service	Value for the `Service` dimension
AWS CloudFormation	`CloudFormation`
AWS CloudHSM	`CloudHSM`
Amazon CloudSearch	`CloudSearch`
AWS CloudShell	`CloudShell`
AWS CloudTrail	`CloudTrail`
Amazon CloudWatch	`CloudWatch`
Amazon CloudWatch Application Signals	`CloudWatch Application Signals`
Amazon CloudWatch Logs	`Logs`
Amazon CloudWatch Application Insights	`CloudWatch Application Insights`
AWS CodeBuild	`CodeBuild`
AWS CodeCommit	`CodeCommit`
Amazon CodeGuru Profiler	`CodeGuru Profiler`
AWS CodePipeline	`CodePipeline`
AWS CodeStar	`CodeStar`
AWS CodeStar Notifications	`CodeStar Notifications`
AWS CodeStar Connections	`CodeStar Connections`
Amazon Cognito Identity pools	`Cognito Identity Pools`
Amazon Cognito Sync	`Cognito Sync`
Amazon Comprehend	`Comprehend`
Amazon Comprehend Medical	`Comprehend Medical`

Service	Value for the Service dimension
AWS Compute Optimizer	ComputeOptimzier
Amazon Connect Customer	Connect
Amazon Connect Customer Customer Profiles	Customer Profiles
AWS Cost and Usage Reports	Cost and Usage Report
AWS Cost Explorer	Cost Explorer
AWS Data Exchange	Data Exchange
AWS Data Lifecycle Manager	Data Lifecycle Manager
AWS Database Migration Service	Database Migration Service
AWS DataSync	DataSync
AWS DeepLens	AWS DeepLens
Amazon Detective	Detective
Device Advisor	Device Advisor
Direct Connect	Direct Connect
AWS Directory Service	Directory Service
DynamoDB Accelerator	DynamoDBAccelerator
Amazon EC2	EC2
EC2 Auto Scaling	EC2 Auto Scaling
Amazon Elastic Container Registry	ECR Public
Amazon Elastic Container Service	ECS
Amazon Elastic File System	EFS

Service	Value for the `Service` dimension
Amazon Elastic Kubernetes Service	`EKS`
AWS Elastic Beanstalk	`Elastic Beanstalk`
Amazon Elastic Inference	`Elastic Inference`
Elastic Load Balancing	`Elastic Load Balancing`
Amazon EMR	`EMR Containers`
AWS Firewall Manager	`Firewall Manager`
Amazon FSx	`FSx`
Amazon GameLift Servers	`GameLift`
AWS Glue DataBrew	`DataBrew`
Amazon Managed Grafana	`Grafana`
AWS IoT Greengrass	`Greengrass`
AWS Ground Station	`Ground Station`
AWS Health APIs And Notifications	`AWS Health APIs And Notifications`
Amazon Interactive Video Service	`IVS`
AWS IoT Core	`IoT`
AWS IoT RoboRunner	`IoT RoboRunner`
AWS IoT SiteWise	`IoT Sitewise`
AWS IoT Wireless	`IoT Wireless`
Amazon Kendra	`Kendra`
Amazon Keyspaces (for Apache Cassandra)	`Keyspaces`

Service	Value for the Service dimension
Amazon Managed Service for Apache Flink	Kinesis Analytics
Amazon Data Firehose	Firehose
Kinesis Video Streams	Kinesis Video Streams
AWS Key Management Service	KMS
AWS Lambda	Lambda
AWS Launch Wizard	Launch Wizard
Amazon Lex	Amazon Lex
Amazon Lightsail	Lightsail
Amazon Location Service	Location
Amazon Machine Learning	Amazon Machine Learning
Amazon Macie	Macie
Amazon Managed Blockchain (AMB) Query	Amazon Managed Blockchain Query
AWS Managed Services	AWS Managed Services
AWS Marketplace Commerce Analytics	Marketplace Analytics Service
AWS Elemental MediaConnect	MediaConnect
AWS Elemental MediaConvert	MediaConvert
AWS Elemental MediaLive	MediaLive
AWS Elemental MediaTailor	MediaTailor
AWS Mobile Hub	Mobile Hub
AWS Network Firewall	Network Firewall

Service	Value for the Service dimension
OpsWorks	OpsWorks
OpsWorks for Configuration Management	OPsWorks CM
AWS Outposts	Outposts
AWS Organizations	Organizations
Amazon RDS Performance Insights	Performance Insights
Amazon Pinpoint	Pinpoint
AWS Private Certificate Authority	Private Certificate Authority
Amazon Managed Service for Prometheus	Prometheus
AWS Proton	Proton
Amazon Quantum Ledger Database (Amazon QLDB)	QLDB
Amazon RDS	RDS
Amazon Redshift	Redshift Data API
Amazon Rekognition	Rekognition
AWS Resource Access Manager	Resource Access Manager
AWS Resource Groups	Resource Groups
AWS Resource Groups Tagging API	Resource Groups Tagging API
Amazon Route 53 Domains	Route 53 Domains
Amazon Route 53 Resolver	Route 53 Resolver
Amazon S3	S3
Amazon Glacier	Amazon S3 Glacier

Service	Value for the Service dimension
Amazon SageMaker Runtime	Sagemaker
Savings Plans	Savings Plans
AWS Secrets Manager	Secrets Manager
AWS Security Hub CSPM	Security Hub
AWS Server Migration Service	AWS Server Migration Service
AWS Service Catalog AppRegistry	Service Catalog AppRegistry
Service Quotas	Service Quotas
AWS Shield	Shield
AWS Signer	Signer
Amazon Simple Notification Service	SNS
Amazon Simple Email Service	SES
Amazon Simple Queue Service	SQS
Identity Store	Identity Store
Storage Gateway	Storage Gateway
AWS Support	Support
Amazon Simple Workflow Service	SWF
Amazon Textract	Textract
AWS IoT Things Graph	ThingsGraph
Amazon Timestream	Timestream
Amazon Transcribe	Transcribe

Service	Value for the Service dimension
Amazon Translate	Translate
Amazon Transcribe streaming transcription	Transcribe Streaming
AWS Transfer Family	Transfer
AWS WAF	WAF
Amazon WorkLink	WorkLink
Amazon WorkMail	Amazon WorkMail
Amazon WorkSpaces	Workspaces
AWS X-Ray	X-Ray

Some services report usage metrics for additional APIs as well. To see whether an API reports usage metrics to CloudWatch, use the CloudWatch console to see the metrics reported by that service in the AWS/Usage namespace.

To see the list of a service's APIs that report usage metrics to CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **Browse** tab, choose **Usage**, and then choose **By AWS Resource**.
4. In the search box near the list of metrics, enter the name of the service. The metrics are filtered by the service you entered.

CloudWatch usage metrics

CloudWatch collects metrics that track the usage of some AWS resources. These metrics correspond to AWS service quotas. Tracking these metrics can help you proactively manage your quotas. For more information, see Visualizing your service quotas and setting alarms. Service quota usage metrics are in the AWS/Usage namespace and are collected every minute.

The metrics that can be published in this namespace include `CallCount`, `ResourceCount`, and `ThrottleCount`. These metrics are published with the dimensions `Resource`, `Service`, and `Type`. The `Resource` dimension specifies the name of the API operation being tracked. For example, the `CallCount` metric with the dimensions `"Service": "CloudWatch"`, `"Type": "API"` and `"Resource": "PutMetricData"` indicates the number of times the CloudWatch `PutMetricData` API operation has been called in your account.

The `CallCount` metric does not have a specified unit. The most useful statistic for the metric is `SUM`, which represents the total operation count for the 1-minute period.

Metrics

Metric	Description
`CallCount`	The number of specified operations performed in your account.

Dimensions

Dimension	Description
`Service`	The name of the AWS service containing the resource. For CloudWatch usage metrics, the value for this dimension is `CloudWatch` .
`Class`	The class of resource being tracked. CloudWatch API usage metrics use this dimension with a value of `None`.
`Type`	The type of resource being tracked. When the `Service` dimension is `CloudWatch` , the only valid value for `Type` is `API`.
`Resource`	The name of the API operation. Valid values include the following: `DeleteAlarms` , `DeleteDashboards` , `DescribeAlarmHistory` , `DescribeAlarms` , `GetDashboard` , `GetMetricData` , `GetMetricStatistics` , `ListMetrics` , `PutDashboard` , and `PutMetricData`

SDK and API reference

This section provides reference material for developers who interact with CloudWatch programmatically. Use AWS SDKs to integrate CloudWatch into your applications, review code examples for common operations, and learn about the protocols and security model that CloudWatch supports.

Topics

- FAQs on CloudWatch supported protocols
- Using CloudWatch with an AWS SDK
- Security in Amazon CloudWatch

FAQs on CloudWatch supported protocols

CloudWatch SDKs started using two additional protocols: AWS JSON 1.0 and Smithy RPC v2 CBOR. These protocols are more performant than AWS Query that was previously utilized by the CloudWatch SDKs and CLIs. AWS SDKs will prioritize the protocol that is the most performant for each language.

What is the AWS JSON protocol, and how does it differ from existing CloudWatch API requests and responses?

JSON is one of the most widely used and accepted data formats for communication between heterogeneous systems. CloudWatch uses JSON as a medium to communicate between an AWS SDK client (for example, Java, Python, Golang, JavaScript) and CloudWatch server. An HTTP request of an CloudWatch API operation accepts input in the form of JSON. The CloudWatch operation is executed and the response of execution is shared back to the SDK client in the form of JSON. Compared to AWS Query, JSON is more efficient at transporting data between client and server.

- CloudWatch AWS JSON protocol acts as a mediator between CloudWatch client and server.
- The CloudWatch AWS SDK handles the serialization (convert object to JSON format) and de-serialization (convert JSON format to object) between the CloudWatch client and server.

What is the Smithy RPC v2 CBOR protocol, and how does it differ from existing CloudWatch API requests and responses?

Smithy RPC v2 CBOR (Concise Binary Object Representation) is a protocol developed by Amazon for more efficient data transmission. It's a data format designed to keep message sizes compact, and allow the format to evolve without requiring version negotiation between systems. The key difference from JSON encoding is that JSON uses human-readable text (like "name": "John"), while CBOR uses a binary format (sequences of 1s and 0s). This binary approach creates smaller data packages that travel faster over the network and reduces serialization time since computers process binary data directly.

- CloudWatch RPC v2 CBOR protocol acts as a mediator between CloudWatch client and server.
- The CloudWatch SDK handles the serialization (convert object to CBOR format) and deserialization (convert CBOR format to object) between the CloudWatch client and server.

How do I get started with the newer protocols for CloudWatch?

To get started, use the latest AWS SDK/CLI version to use the latest supported protocol for CloudWatch. Upgrade your AWS SDK to the specified version or any subsequent version.

Language	Protocol	SDK versions
C++	RPC v2 CBOR	1.11.708
Golang 2.x	RPC v2 CBOR	1.53.0
Java 1.x	RPC v2 CBOR	1.12.795
Java 2.x	RPC v2 CBOR	2.40.6
JavaScript v3.x	AWS JSON 1.0	3.949.0
.NET v3	AWS JSON 1.0	3.7.1182.0
.NET v4	RPC v2 CBOR	4.0.150.0
PHP	AWS JSON 1.0	3.367.0

Language	Protocol	SDK versions
Python-boto3	AWS JSON 1.0	1.42.7
Python-botocore	AWS JSON 1.0	1.42.7
AWS CLI v1	AWS JSON 1.0	1.43.13
AWS CLI v2	AWS JSON 1.0	2.32.14
Ruby	AWS JSON 1.0	1.126.0
Powershell	AWS JSON 1.0	5.0.114
Rust	RPC v2 CBOR	1.100.0
Swift	RPC v2 CBOR	1.6.14
Kotlin	RPC v2 CBOR	1.5.100

What are the risks of enabling these new protocols for my CloudWatch workloads?

The migration into these new SDK versions is expected to be fully transparent and effortless for customers. There is one known behavioral change in between protocols related with how `null` values are treated within lists: CloudWatch API calls with list inputs that may contain `null` values will be handled differently in the AWS JSON and Smithy RPC v2 CBOR protocols as compared to the previous AWS Query protocol: In AWS Query protocol, null values within lists are removed, while in JSON/CBOR protocols will be propagated through the API calls.

Additionally, if you are using a custom implementation of AWS SDK or a combination of custom clients and AWS SDK to interact with CloudWatch that generates AWS Query based (aka XML-based) responses, it may be incompatible with AWS JSON or CBOR protocol. If you encounter any issues, contact AWS Support.

What if I am already on the latest AWS SDK version, but my open sourced solution does not support JSON or CBOR?

You must change your SDK version to a previous version. See the How do I get started with the newer protocols for CloudWatch? section and select a version lower than listed. If you change your AWS SDK to the previous version, your CloudWatch APIs will use the AWS Query protocol.

What regions are supported for AWS JSON and CBOR protocols used in CloudWatch APIs

CloudWatch supports AWS JSON and CBOR protocol in all AWS regions where CloudWatch is available.

What latency improvements can I expect when upgrading to the specified AWS SDK versions for CloudWatch using the newer protocols?

Both JSON and CBOR provide up to 80% lower latency and CPU usage when compared to the AWS Query. Both protocols are up to 20% more efficient in network bandwidth usage.

Will the AWS Query protocol be deprecated?

AWS Query protocol will continue to be supported. You can continue using the AWS Query protocol as long as your AWS SDK version is set any version before what is listed in How do I get started with the newer protocols for CloudWatch?.

Where can I find more information about AWS JSON and Smithy RPC v2 CBOR protocols?

You can find more information about JSON protocol at AWS JSON 1.0 protocol in the Smithy documentation, and about CBOR at AWS RPC v2 CBOR protocol in the Smithy documentation.

For more about CloudWatch API requests, see Making API Requests in the CloudWatch API Reference Guide.

Using CloudWatch with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples
AWS SDK for Python v4	AWS SDK for Python v4 code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to CloudWatch, see Code examples for CloudWatch using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Security in Amazon CloudWatch

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to CloudWatch, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon CloudWatch. It shows you how to configure Amazon CloudWatch to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudWatch resources.

Contents

- Data protection in Amazon CloudWatch
- Encryption at rest for OpenTelemetry metrics
- Identity and access management for Amazon CloudWatch
- Compliance validation for Amazon CloudWatch
- Resilience in Amazon CloudWatch
- Infrastructure security in Amazon CloudWatch

- AWS Security Hub
- Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints
- Security considerations for Synthetics canaries

Data protection in Amazon CloudWatch

The AWS shared responsibility model applies to data protection in Amazon CloudWatch. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with CloudWatch or other AWS services using the console, API, AWS CLI, or AWS

SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

CloudWatch always encrypts data at rest. By default, CloudWatch uses AWS owned keys to encrypt your data. You don't need to take any action to use AWS owned keys.

For CloudWatch Dataset resources, you can optionally use a customer managed key in AWS Key Management Service to encrypt your data. For more information, see Encryption at rest for OpenTelemetry metrics.

Encryption in transit

CloudWatch uses end-to-end encryption of data in transit. For more information, see Encryption of data in transit.

Encryption at rest for OpenTelemetry metrics

What is a CloudWatch Dataset

OpenTelemetry (OTel) metrics that you send to Amazon CloudWatch are stored in a resource called a Dataset. Every AWS account has a `default` dataset in each Region where all OTel metrics reside. The `default` dataset is the only supported dataset — you cannot create additional datasets.

Datasets can be encrypted and tagged like other AWS resources. The dataset ARN has the following format:

`arn:`*`{partition}`*`:cloudwatch:`*`{region}`*`:`*`{account-id}`*`:dataset/default`

To view the current encryption configuration of your dataset, use the `GetDataset` API:

```
aws cloudwatch get-dataset \
    --dataset-identifier default
```

If a customer managed key is associated with the dataset, the response includes the key ARN. If no customer managed key is associated, the dataset is encrypted with an AWS owned key.

Options for encryption at rest

CloudWatch always encrypts Dataset data at rest. By default, CloudWatch encrypts all data at rest using AWS owned keys. You don't need to take any action to protect your data using AWS owned keys. For more information, see AWS owned keys in the AWS Key Management Service Developer Guide.

If you want to manage the keys that are used to encrypt your Dataset data, you can use a customer managed key in AWS Key Management Service (AWS KMS). For more information, see Customer managed keys in the AWS Key Management Service Developer Guide.

When you use a customer managed key, AWS KMS charges apply. For more information about pricing, see AWS Key Management Service pricing.

How CloudWatch uses a customer managed key for Dataset encryption

> **Important**
>
> Customer managed key encryption applies to the `default` dataset. The `default` dataset is the only supported dataset — you cannot create additional datasets.

When you associate a customer managed key with the `default` dataset, CloudWatch uses the key to encrypt all OTel metric data stored in that dataset.

CloudWatch uses the service principal (`cloudwatch.amazonaws.com`) directly with key policy permissions. CloudWatch does not use grants or IAM roles to access your AWS KMS key.

CloudWatch does not cache data keys. However, CloudWatch caches `kms:Decrypt` responses for up to 15 minutes. Changes to a key policy might take up to 15 minutes to take effect.

CloudWatch uses the following encryption context for all AWS KMS cryptographic operations:

- Key: `aws:cloudwatch:arn`
- Value: `arn:`*`{partition}`*`:cloudwatch:`*`{region}`*`:`*`{account-id}`*`:dataset/default`

Configuring a customer managed key for Dataset

The AWS KMS key that you use with CloudWatch Dataset must meet the following requirements:

- The key must be a symmetric encryption key (SYMMETRIC_DEFAULT) with key usage ENCRYPT_DECRYPT. Asymmetric keys are not supported.
- Multi-Region keys are not supported.
- The key must be in the same AWS Region as the Dataset.
- You must specify the key as a fully qualified key ARN. Key aliases and key IDs are not supported.

Configuring key policy permissions

To use a customer managed key with CloudWatch Dataset, the key policy must grant CloudWatch permission to use the key. The following example key policy grants CloudWatch the necessary permissions and includes confused deputy protection.

The caller who associates or uses the Dataset must have `kms:Decrypt` permission, scoped to the CloudWatch `ViaService` and encryption context as shown in the `AllowCallerDecrypt` statement below. Replace *`YourApplicationRole`* with the IAM role used to call CloudWatch Dataset APIs.

Example Key policy for CloudWatch Dataset encryption

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudWatchDatasetDescribeKey",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudwatch.amazonaws.com"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
                }
            }
        },
```

```
        {
            "Sid": "AllowCloudWatchDatasetEncryption",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudwatch.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
                }
            }
        },
        {
            "Sid": "AllowCallerDecrypt",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/YourApplicationRole"
            },
            "Action": "kms:Decrypt",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "cloudwatch.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
                }
            }
        }
    ]
}
```

Replace the example account ID (111122223333) and the example AWS Region (us-east-1) with your own values.

For more information about key policies, see Key policies in AWS KMS in the AWS Key Management Service Developer Guide.

To associate a customer managed key with a Dataset

Use the AssociateDatasetKmsKey API to associate a customer managed key with a Dataset. You must specify default as the dataset identifier.

To associate a customer managed key by using the AWS CLI, run the following command:

```
aws cloudwatch associate-dataset-kms-key \
    --dataset-identifier default \
    --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

Changing or removing encryption configuration

You can change or remove the customer managed key that encrypts your Dataset data.

To change the customer managed key

To replace the customer managed key, call AssociateDatasetKmsKey again with a new key ARN. The caller must have kms:Decrypt permission on both the current key and the new key. CloudWatch begins using the new key for subsequent encryption operations.

> **Note**
>
> If the currently associated key has been deleted, is scheduled for deletion, is pending import, is unavailable, or has been disabled, CloudWatch does not require kms:Decrypt permission on the current key and the rotation proceeds. If the key was only disabled, consider re-enabling it rather than rotating, because re-enabling allows CloudWatch to resume decrypting existing metric data encrypted with that key.

To remove the customer managed key

To remove the customer managed key and revert to AWS owned key encryption, call DisassociateDatasetKmsKey. The caller must have kms:Decrypt permission on the currently associated key.

```
aws cloudwatch disassociate-dataset-kms-key \
    --dataset-identifier default
```

Note

If the currently associated key has been deleted, is scheduled for deletion, is pending import, is unavailable, or has been disabled, CloudWatch does not require `kms:Decrypt` permission on that key and the disassociation proceeds. If the key was only disabled, consider re-enabling it instead of disassociating, because re-enabling allows CloudWatch to resume decrypting your existing metric data.

Important

After you disassociate a customer managed key, there is a 3-hour enforcement window during which CloudWatch still requires `kms:Decrypt` permission on the previously associated key. Don't disable or delete the key during this window.

Recovering from a deleted or unusable KMS key

If the customer managed KMS key associated with your dataset is deleted, scheduled for deletion, or becomes unavailable, your dataset enters a degraded state:

- CloudWatch can't encrypt new metric data that you publish to the dataset, and ingestion fails.
- CloudWatch can't decrypt existing data that was encrypted with the unusable key, so query operations fail.

Your recovery options depend on the key state:

- **Key is disabled** — Re-enable the key in AWS KMS. This is the preferred recovery path because CloudWatch can resume decrypting your existing metric data after the key is active again.
- **Key is pending deletion** — Cancel the key deletion in AWS KMS before the waiting period expires. After the key returns to an enabled state, CloudWatch resumes normal operations.
- **Key has been permanently deleted or is otherwise unrecoverable** — Remove the stale association or rotate to a new key (see below). Note that existing data encrypted with the deleted key is permanently unreadable.

If the key is permanently deleted or otherwise unrecoverable, you can remove the association or rotate to a new key without needing access to the old key:

Remove the stale key association

Call `DisassociateDatasetKmsKey` to remove the association with the unusable key. CloudWatch reverts to AWS owned key encryption for new data. Existing data that was encrypted with the deleted key is no longer readable.

```
aws cloudwatch disassociate-dataset-kms-key \
    --dataset-identifier default
```

Rotate to a new key

Call `AssociateDatasetKmsKey` with a new, valid KMS key ARN. CloudWatch validates the new key and associates it with your dataset. Access to the previously associated key is not required. Existing data that was encrypted with the deleted key is no longer readable, but new data is encrypted with the new key.

```
aws cloudwatch associate-dataset-kms-key \
    --dataset-identifier default \
    --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/new-key-id
```

Error responses for unusable keys

When you call `DisassociateDatasetKmsKey` or `AssociateDatasetKmsKey` (rotation path) and the currently associated key is inaccessible, CloudWatch does not require access to that key and the operation proceeds normally. This applies when:

- The key has been deleted (KMS returns `NotFoundException`).
- The key is in the `PendingDeletion`, `PendingImport`, or `Unavailable` state.
- The key has been disabled.

If you call these operations when the key is accessible but the caller lacks `kms:Decrypt` permission, the operation fails with one of the following errors. Each error message includes the KMS request ID for troubleshooting.

`ResourceNotFoundException`

The specified KMS key does not exist. Message format: `KMS key key-arn does not exist. KMS request id: request-id`.

`ValidationException`

The specified KMS key is in an unusable state that is not tolerated on this code path. Message format: `KMS key key-arn is in an unusable state: state-description. KMS request id: request-id`.

`AccessDeniedException`

The specified KMS key is disabled, or the caller lacks permission. Message format: `KMS key key-arn is disabled. KMS request id: request-id`.

For more information about KMS key states, see Key states of AWS KMS keys in the *AWS Key Management Service Developer Guide*.

Scoping down key policy access

You can use conditions in the key policy to limit access to your AWS KMS key.

Encryption context condition

Use the `kms:EncryptionContext:aws:cloudwatch:arn` condition key to restrict key usage to your `default` Dataset.

```
"Condition": {
    "StringEquals": {
        "kms:EncryptionContext:aws:cloudwatch:arn": "arn:aws:cloudwatch:us-
east-1:111122223333:dataset/default"
    }
}
```

Confused deputy protection

Use the `aws:SourceArn` and `aws:SourceAccount` conditions to prevent cross-account confused deputy attacks.

```
"Condition": {
    "StringEquals": {
```

```
        "aws:SourceAccount": "111122223333"
    },
    "ArnLike": {
        "aws:SourceArn": "arn:aws:cloudwatch:us-east-1:111122223333:dataset/default"
    }
}
```

kms:ViaService condition

Use the `kms:ViaService` condition key to restrict key usage to requests that come from CloudWatch.

```
"Condition": {
    "StringEquals": {
        "kms:ViaService": "cloudwatch.us-east-1.amazonaws.com"
    }
}
```

Monitoring CloudWatch interaction with AWS KMS

You can use AWS CloudTrail to track the requests that CloudWatch sends to AWS KMS on your behalf. The AWS CloudTrail log entries use the service principal `cloudwatch.amazonaws.com` and a `ViaService` value of `cloudwatch.{region}.amazonaws.com`.

The following CloudTrail event names appear in log entries for CloudWatch Dataset encryption operations:

- `GenerateDataKey`
- `Encrypt`
- `Decrypt`
- `DescribeKey`
- `ReEncrypt`

Each log entry includes the encryption context, which you can use to identify the specific Dataset that the operation applies to.

For more information about monitoring AWS KMS key usage, see Monitoring AWS Key Management Service in the AWS Key Management Service Developer Guide.

Identity and access management for Amazon CloudWatch

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CloudWatch resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon CloudWatch works with IAM
- Identity-based policy examples for Amazon CloudWatch
- Troubleshooting Amazon CloudWatch identity and access
- CloudWatch dashboard permissions update
- AWS managed (predefined) policies for CloudWatch
- Customer managed policy examples
- Using condition keys to limit access to CloudWatch
- Using service-linked roles for CloudWatch
- Using service-linked roles for CloudWatch RUM
- Using service-linked roles for CloudWatch Application Insights
- AWS managed policies for Amazon CloudWatch Application Insights
- Amazon CloudWatch permissions reference

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon CloudWatch identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon CloudWatch works with IAM)

- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon CloudWatch)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon CloudWatch works with IAM

Before you use IAM to manage access to CloudWatch, learn what IAM features are available to use with CloudWatch.

The following tables list the IAM features that you can use with Amazon CloudWatch.

IAM feature	CloudWatch support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how CloudWatch and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for CloudWatch

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for CloudWatch

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Resource-based policies within CloudWatch

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for CloudWatch

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CloudWatch actions, see Actions defined by Amazon CloudWatch in the *Service Authorization Reference*.

Policy actions in CloudWatch use the following prefix before the action:

```
cloudwatch
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "cloudwatch:action1",
      "cloudwatch:action2"
         ]
```

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy resources for CloudWatch

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of CloudWatch resource types and their ARNs, see Resources defined by Amazon CloudWatch in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon CloudWatch.

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

Policy condition keys for CloudWatch

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match

the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of CloudWatch condition keys, see Condition keys for Amazon CloudWatch in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon CloudWatch.

To view examples of CloudWatch identity-based policies, see Identity-based policy examples for Amazon CloudWatch.

ACLs in CloudWatch

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CloudWatch

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with CloudWatch

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for CloudWatch

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for CloudWatch

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break CloudWatch functionality. Edit service roles only when CloudWatch provides guidance to do so.

Identity-based policy examples for Amazon CloudWatch

By default, users and roles don't have permission to create or modify CloudWatch resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by CloudWatch, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon CloudWatch in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the CloudWatch console

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CloudWatch resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the CloudWatch console

To access the Amazon CloudWatch console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the CloudWatch resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To make sure that users and roles can still use the CloudWatch console, also attach the CloudWatch `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Permissions needed for CloudWatch console

The full set of permissions required to work with the CloudWatch console are listed below. These permissions provide full write and read access to the CloudWatch console.

- application-autoscaling:DescribeScalingPolicies
- autoscaling:DescribeAutoScalingGroups
- autoscaling:DescribePolicies
- cloudtrail:DescribeTrails
- cloudwatch:DeleteAlarms
- cloudwatch:DescribeAlarmHistory
- cloudwatch:DescribeAlarms
- cloudwatch:GetMetricData
- cloudwatch:GetMetricStatistics
- cloudwatch:ListMetrics
- cloudwatch:PutMetricAlarm
- cloudwatch:PutMetricData
- ec2:DescribeInstances

- ec2:DescribeTags
- ec2:DescribeVolumes
- es:DescribeElasticsearchDomain
- es:ListDomainNames
- events:DeleteRule
- events:DescribeRule
- events:DisableRule
- events:EnableRule
- events:ListRules
- events:PutRule
- iam:AttachRolePolicy
- iam:CreateRole
- iam:GetPolicy
- iam:GetPolicyVersion
- iam:GetRole
- iam:ListAttachedRolePolicies
- iam:ListRoles
- kinesis:DescribeStream
- kinesis:ListStreams
- lambda:AddPermission
- lambda:CreateFunction
- lambda:GetFunctionConfiguration
- lambda:ListAliases
- lambda:ListFunctions
- lambda:ListVersionsByFunction
- lambda:RemovePermission
- logs:CancelExportTask
- logs:CreateExportTask
- logs:CreateLogGroup
- logs:CreateLogStream

- logs:DeleteLogGroup
- logs:DeleteLogStream
- logs:DeleteMetricFilter
- logs:DeleteRetentionPolicy
- logs:DeleteSubscriptionFilter
- logs:DescribeExportTasks
- logs:DescribeLogGroups
- logs:DescribeLogStreams
- logs:DescribeMetricFilters
- logs:DescribeQueries
- logs:DescribeSubscriptionFilters
- logs:FilterLogEvents
- logs:GetLogGroupFields
- logs:GetLogRecord
- logs:GetLogEvents
- logs:GetQueryResults
- logs:PutMetricFilter
- logs:PutRetentionPolicy
- logs:PutSubscriptionFilter
- logs:StartQuery
- logs:StopQuery
- logs:TestMetricFilter
- s3:CreateBucket
- s3:ListBucket
- sns:CreateTopic
- sns:GetTopicAttributes
- sns:ListSubscriptions
- sns:ListTopics
- sns:SetTopicAttributes
- sns:Subscribe

- sns:Unsubscribe
- sqs:GetQueueAttributes
- sqs:GetQueueUrl
- sqs:ListQueues
- sqs:SetQueueAttributes
- swf:CreateAction
- swf:DescribeAction
- swf:ListActionTemplates
- swf:RegisterAction
- swf:RegisterDomain
- swf:UpdateAction

Additionally, to view the X-Ray Trace Map, you need `AWSXrayReadOnlyAccess`

Troubleshooting Amazon CloudWatch identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CloudWatch and IAM.

Topics

- I am not authorized to perform an action in CloudWatch
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my CloudWatch resources

I am not authorized to perform an action in CloudWatch

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `cloudwatch:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cloudwatch:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `cloudwatch:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CloudWatch.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CloudWatch. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my CloudWatch resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CloudWatch supports these features, see How Amazon CloudWatch works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

CloudWatch dashboard permissions update

On May 1, 2018, AWS changed the permissions required to access CloudWatch dashboards. Dashboard access in the CloudWatch console now requires permissions that were introduced in 2017 to support dashboard API operations:

- **cloudwatch:GetDashboard**
- **cloudwatch:ListDashboards**
- **cloudwatch:PutDashboard**
- **cloudwatch:DeleteDashboards**

To access CloudWatch dashboards, you need one of the following:

- The **AdministratorAccess** policy.
- The **CloudWatchFullAccess** policy.
- A custom policy that includes one or more of these specific permissions:
  - `cloudwatch:GetDashboard` and `cloudwatch:ListDashboards` to be able to view dashboards
  - `cloudwatch:PutDashboard` to be able to create or modify dashboards
  - `cloudwatch:DeleteDashboards` to be able to delete dashboards

For more information about using policies to change permissions for an IAM user, see Changing Permissions for an IAM user.

For more information about CloudWatch permissions, see Amazon CloudWatch permissions reference.

For more information about dashboard API operations, see PutDashboard in the Amazon CloudWatch API Reference.

AWS managed (predefined) policies for CloudWatch

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS managed policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to CloudWatch.

Topics

- CloudWatch updates to AWS managed policies
- CloudWatchFullAccessV2
- CloudWatchFullAccess
- CloudWatchReadOnlyAccess
- CloudWatchActionsEC2Access
- CloudWatch-CrossAccountAccess
- CloudWatchAutomaticDashboardsAccess
- CloudWatchAgentServerPolicy
- CloudWatchAgentAdminPolicy
- CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy
- AWS managed (predefined) policies for CloudWatch cross-account observability
- AWS managed (predefined) policies for CloudWatch investigations
- AWS managed (predefined) policies for CloudWatch Application Signals
- AWS managed (predefined) policies for CloudWatch Synthetics
- AWS managed (predefined) policies for Amazon CloudWatch RUM
- AWS managed policy for AWS Systems Manager Incident Manager

CloudWatch updates to AWS managed policies

View details about updates to AWS managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
CloudWatchNetworkMonitorServiceRolePolicy – Updated policy	The **CloudWatchNetworkMonitorServiceRolePolicy** managed policy now includes the `ec2:DescribeVpcPeeringConnections` and `ec2:DescribeVpcEndpoints` permissions. Network Synthetic Monitor uses these permissions to identify the destination Region reached across an inter-Region VPC peering connection or an interface VPC endpoint. It then generates network health indicator (NHI) values for the path to that destination Region.	October 28, 2026
CloudWatchFullAccessV2 – Updated policy	CloudWatch added the `kms:DescribeKey` permission to **CloudWatchFullAccessV2**. CloudWatch Logs requires this permission from the caller to create a log group with a customer managed AWS	October 20, 2026

Change	Description	Date
	KMS key or to associate a key with a log group. You also need it to encrypt the `aws/spans` log group with a customer managed key when you enable Transaction Search. You can use this permission only for requests that CloudWatch Logs makes on your behalf (`kms:ViaService`), and only for keys in your own account.	

Change	Description	Date
CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy – Updated policy	CloudWatch updated the **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** to include additional permissions. `elasticloadbalancing:` permissions describe load balancers, listeners, rules, target groups, and target health. `networkmanager:` permissions describe AWS Cloud WAN core networks, attachments, and routes. `config:` and `iam:` permissions create, delete, and describe a service-linked configuration recorder and create the AWS Config service-linked role. With these permissions, Network Flow Monitor can generate topology snapshots of Elastic Load Balancing and AWS Cloud WAN resources and record resource configuration data through AWS Config.	August 31, 2026

Change	Description	Date
CloudWatchNetworkMonitorServiceRolePolicy – Updated policy	We added the `ec2:DescribeTransitGatewayPeeringAttachments` permission to the **CloudWatchNetworkMonitorServiceRolePolicy** managed policy. Network Synthetic Monitor uses this permission to identify the destination Region reached across an AWS Transit Gateway peering attachment. It then generates network health indicator (NHI) values for the path to that destination Region.	August 28, 2026
CloudWatchAutomaticDashboardsAccess – Updated policy	CloudWatch added the `lambda:GetFunctionConfiguration` and `lambda:ListTags` permissions to the `CloudWatchAutomaticDashboardsAccess` managed policy to enable automatic dashboards to retrieve Lambda function configuration details and resource tags.	August 6, 2026

Change	Description	Date
CloudWatchFullAccessV2 – Updated policy	CloudWatch added permissions to **CloudWatchFullAccessV2**. Two `iam:PassRole` permissions were added to support the `PutLogAlarm` API: passing a Scheduled Query Execution Role to `logs.amazonaws.com` (scoped to scheduled-query resources), and passing a Log Lines Role to `cloudwatch.amazonaws.com` (scoped to alarm resources).	June 15, 2026
CloudWatchSyntheticsFullAccess – Updated policy	CloudWatch updated the **CloudWatchSyntheticsFullAccess** policy. The `cloudwatch:ListMetrics` permission was added so that CloudWatch Synthetics can list available metrics. Additionally, the `apigateway:GET` permission has been changed from allowing all resources to specific API Gateway resources: REST APIs, REST API stages, REST API stage Swagger exports, and HTTP APIs.	March 31, 2026

Change	Description	Date
AIOpsAssistantPolicy – Updated policy	CloudWatch updated the **AIOpsAssistantPolicy** IAM policy. It added permissions to enable CloudWatch investigations to assist in queries, troubleshooting, and topology mapping. The following permissions were added: `appsync:GetGraphqlApiEnvironmentVariables`, `cloudtrail:GetEventConfiguration`, `kms:GetKeyPolicy`, and `s3:GetBucketAbac`	February 06, 2026
AIOpsAssistantPolicy – Updated policy	CloudWatch updated the **AIOpsAssistantPolicy** IAM policy. It added permissions to enable CloudWatch investigations to assist in queries, troubleshooting, and topology mapping. The following permissions were added: `kms:GetKeyRotationStatus`, `kms:ListAliases`, and `kms:ListKeyRotations`	December 17, 2025

Change	Description	Date
CloudWatchNetworkMonitorServiceRolePolicy – Updated policy	CloudWatch added the `ec2:DescribeRouteTables` , `ec2:DescribeTransitGatewayAttachments` , `ec2:DescribeTransitGatewayRouteTables` , `ec2:SearchTransitGatewayRoutes` permissions to the **CloudWatchNetworkMonitorServiceRolePolicy** managed policy to grant permissions for Network Synthetic Monitor to generate the Network Health Indicator values for customers using Transit Gateways.	December 12, 2025
CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy – Updated policy	CloudWatch added the `ec2:DescribeVpcEndpoints` and `ec2:DescribeVpcEndpointServiceConfigurations` permissions to the **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** managed policy to grant permissions for Network Flow Monitor to generate topology snapshots of VPC endpoints and vpce service config.	December 10, 2025

Change	Description	Date
CloudWatchFullAccessV2 – Updated policy	CloudWatch added permissions to **CloudWatchFullAccessV2**. Permissions for observability administration actions were added to allow full access to telemetry pipelines and S3 table integrations.	December 02, 2025
CloudWatchReadOnlyAccess – Updated policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. Permissions for observability administration actions were added to allow read-only access to telemetry pipelines and S3 table integrations.	December 02, 2025
CloudWatchFullAccessV2 – Updated policy	CloudWatch updated the **CloudWatchFullAccessV2** policy to include permissions to support viewing change events for a given entity and time range, and querying uninstrumented services and resources from Application Signals by enabling resource explorer.	November 20th, 2025

Change	Description	Date
CloudWatchApplicationSignalsFullAccess – Updated policy	CloudWatch updated the **CloudWatchApplicationSignalsFullAccess** policy to include permissions to support viewing change events for a given entity and time range, and querying uninstrumented services and resources from Application Signals by enabling resource explorer.	November 20th, 2025
CloudWatchReadOnlyAccess – Updated policy	CloudWatch updated the **CloudWatchReadOnlyAccess** policy to include permissions to support viewing change events for a given entity and time range, and querying uninstrumented services and resources from Application Signals.	November 20th, 2025
CloudWatchApplicationSignalsReadOnlyAccess – Updated policy	CloudWatch updated the **CloudWatchApplicationSignalsReadOnlyAccess** policy to include permissions to support viewing change events for a given entity and time range, and querying uninstrumented services and resources from Application Signals.	November 20th, 2025

Change	Description	Date
CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy	CloudWatch updated the policy named **CloudWatchApplicationSignalsServiceRolePolicy**. Updated the policy to include `resource-explorer-2:Search` and `cloudtrail:CreateServiceLinkedChannel` to enable new Application Signals features.	November 20, 2025
AIOpsConsoleAdminPolicy – Updated policy	CloudWatch updated the **AIOpsConsoleAdminPolicy** policy to include Amazon Q integration permissions, enabling users to interact with CloudWatch investigations incident reports through Amazon Q's conversational interface.	November 17, 2025
AIOpsOperatorAccess – Updated policy	CloudWatch updated the **AIOpsOperatorAccess** policy to include Amazon Q integration permissions, enabling users to interact with CloudWatch investigations incident reports through Amazon Q's conversational interface.	November 7, 2025

Change	Description	Date
CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy – Updated policy	CloudWatch added the `ec2:DescribeManagedPrefixLists` and `ec2:GetManagedPrefixListEntries` permissions to the **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** managed policy to grant permissions for Network Flow Monitor to generate topology snapshots of managed prefix lists.	November 06, 2025
AIOpsAssistantIncidentReportPolicy – New policy	CloudWatch added the **AIOpsAssistantIncidentReportPolicy** policy to enable CloudWatch investigations to generate incident reports from investigation data, including permissions to access investigations, create reports, and manage AI-derived facts.	October 10, 2025
AIOpsOperatorAccess – Updated policy	CloudWatch updated the **AIOpsOperatorAccess** policy to include incident report generation permissions, allowing users to create and manage incident reports and work with AI-derived facts in CloudWatch investigations.	October 10, 2025

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy.	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. Permissions for observability administration actions were added to allow read-only access to telemetry rules, centralization configurations, and resource telemetry data across AWS Organizations.	October 10, 2025
CloudWatchFullAccessV2 – Updates to existing policies	CloudWatch updated **CloudWatchFullAccessV2** to include CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals.	October 8, 2025
CloudWatchReadOnlyAccess – Updates to existing policies	CloudWatch updated **CloudWatchReadOnlyAccess** to include CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals.	October 8, 2025

Change	Description	Date
CloudWatchApplicationSignalsReadOnlyAccess – Updated policy	CloudWatch updated the **CloudWatchApplicationSignalsReadOnlyAccess** policy to view how resources and services change in your account and view top observations for service anomalies in your account.	September 29, 2025
CloudWatchApplicationSignalsFullAccess – Updated policy	CloudWatch updated the **CloudWatchApplicationSignalsFullAccess** policy to view how resources and services change in your account and view top observations for service anomalies in your account.	September 29, 2025
AIOpsAssistantPolicy – Updated policy	CloudWatch updated the **AIOpsAssistantPolicy** to allow CloudWatch investigations access to additional resources to assist in queries, troubleshooting, and topology mapping. This policy grants CloudWatch investigations the necessary permissions for conducting investigations.	September 24, 2025

Change	Description	Date
AIOpsConsoleAdminPolicy – Updated policy	CloudWatch updated the **AIOpsConsoleAdminPolicy** policy to permit validation of investigation groups across accounts. This policy grants users access to CloudWatch investigations actions, and to additional AWS actions that are necessary for accessing investigation events.	June 13, 2025
AIOpsOperatorAccess – Updated policy	CloudWatch updated the **AIOpsOperatorAccess** policy to permit validation of investigation groups across accounts. This policy grants users access to CloudWatch investigations actions, and to additional AWS actions that are necessary for accessing investigation events.	June 13, 2025

Change	Description	Date
AIOpsAssistantPolicy – Updates to existing policy	CloudWatch updated the **AIOpsAssistantPolicy** IAM policy. It added permissio ns to enable CloudWatch Application Insights operation al investigations to find information in your resources during investigations. The following permissions were added: `appsync:G etGraphqlApi` , `appsync:GetDataSou rce` , `iam:ListA ttachedRolePolicie s` , `iam:ListRolePolici es` ,and `iam:ListRoles` Additionally, the `elastic-i nference:Describe*` permission was removed from the policy.	June 13, 2025

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	CloudWatch added permissions to the **AmazonCloudWatchRUMReadOnlyAccess** policy. The `synthetics:describeCanaries` and `synthetics:describeCanariesLastRun` was added so that CloudWatch RUM can display associated Synthetics Canaries to the RUM app monitor. The `cloudwatch:GetMetricData` was added so that CloudWatch RUM can display associated CloudWatch metrics to the RUM app monitor. The `cloudwatch:DescribeAlarms` was added so that CloudWatch RUM can display associated CloudWatch alarms to the RUM app monitor. The `logs:DescribeLogGroups` was added so that CloudWatch RUM can display associated CloudWatch logs to the RUM app monitor.	June 28, 2025

Change	Description	Date
	The `xray:GetTraceSummaries` was added so that CloudWatch RUM can display associated X-Ray Trace segments to the RUM app monitor. The `rum:ListTagsForResources` was added so that CloudWatch RUM can display associated tags to the RUM app monitor.	
AIOpsReadOnlyAccess – Updated policy	CloudWatch updated the **AIOpsReadOnlyAccess** policy to permit validation of investigation groups across accounts. This policy grants a user read-only permissions for Amazon AI Operations and other related services.	June 5, 2025
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	CloudWatch added a permission to the **AmazonCloudWatchRUMReadOnlyAccess** policy. The `rum:GetResourcePolicy` permission was added so that CloudWatch RUM can view the resource policy attached to the RUM application monitor.	April 28, 2025

Change	Description	Date
AIOpsConsoleAdminPolicy – New policy	CloudWatch created a new policy named **AIOpsConsoleAdminPolicy**. This policy grants users full administrative access for managing CloudWatch investigations, including the management of trusted identity propagation, and the management of IAM Identity Center and organizational access.	December 3, 2024
AIOpsOperatorAccess – New policy	CloudWatch created a new policy named **AIOpsOperatorAccess**. This policy grants users access to CloudWatch investigations actions, and to additional AWS actions that are necessary for accessing investigation events.	December 3, 2024
AIOpsReadOnlyAccess – New policy	CloudWatch created a new policy named **AIOpsReadOnlyAccess**. This policy grants a user read-only permissions for Amazon AI Operations and other related services.	December 3, 2024

Change	Description	Date
AIOpsAssistantPolicy – New policy	CloudWatch created a new policy named **AIOpsAssistantPolicy**. You don't assign this policy to a user. You assign this policy to the Amazon AI Operations assistant to enable CloudWatch investigations to analyze your AWS resources during the investigation of operational events.	December 3, 2024
CloudWatchFullAccessV2 – Updates to existing policies	CloudWatch updated both **CloudWatchFullAccessV2** and **CloudWatchFullAccess**. Permissions for Amazon OpenSearch Service were added to to enable CloudWatch Logs integration with OpenSearch Service for some features.	December 1, 2024

Change	Description	Date
CloudWatchNetworkFlowMonitorServiceRolePolicy – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorServiceRolePolicy**. The **CloudWatchNetworkFlowMonitorServiceRolePolicy** grants permissions for Network Flow Monitor to publish metrics to CloudWatch. It also allows the service to use AWS Organizations to get information for multi-account scenarios.	December 1, 2024
CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy**. The **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** grants permissions for Network Flow Monitor to generate topology snapshots of resources used in your account.	December 1, 2024

Change	Description	Date
CloudWatchNetworkFlowMonitorAgentPublishPolicy – New policy	CloudWatch added a new policy **CloudWatchNetworkFlowMonitorAgentPublishPolicy**. The **CloudWatchNetworkFlowMonitorAgentPublishPolicy** grants permissions for resources, such as Amazon EC2 and Amazon EKS instances, to send telemetry reports (metrics) to a Network Flow Monitor endpoint.	December 1, 2024

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch updated the policy named **CloudWatchSyntheticsFullAccess**. The following CloudWatch Logs actions have been added to allow CloudWatch Synthetics to get and use canary log data in Lambda log groups. The `lambda:GetFunction` permission has also been added to allow Synthetics to get information about a specific function. • `logs:GetLogRecord` – Required to expand a log entry in CloudWatch Logs Insights. (Required to view logs in the Synthetics console.) • `logs:DescribeLogStreams` – Required to list all log streams on a log group. • `logs:StartQuery` – Required to start a Logs Insights query (required in the Synthetics console to view logs). • `logs:GetLogEvents` –	November 20, 2024

Change	Description	Date
	Required to list log events from the specified log stream. Used for querying a specific log stream on a log group. • `logs:FilterLogEvents` – Required to view logs in a canary's log group. • `logs:GetLogGroupFields` – Required for running a Logs Insights query in the console. (The Synthetics console links to the Logs Insights query. Without this permission, Logs Insights queries on a log group fail.) Additionally, Lambda layer version actions now apply to all CloudWatch Synthetics layer ARNs.	

Change	Description	Date
CloudWatchInternetMonitorReadOnlyAccess – New **CloudWatchInternetMonitorReadOnlyAccess**. This policy grants read only access to resources and actions available in the CloudWatch console for Internet Monitor. The scope of this policy includes `internetmonitor:` so that users can use read-only Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch metrics. It includes some `logs:` policies to manage log queries.	November 14, 2024	

Change	Description	Date
CloudWatchInternetMonitorFullAccess – New policy	CloudWatch created a new policy named **CloudWatchInternetMonitorFullAccess**. This policy grants full access to resources and actions available in the CloudWatch console for Internet Monitor. The scope of this policy includes `internetmonitor:` so that users can use Internet Monitor actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `ec2:`, `cloudfront:`, `elasticloadbalancing:`, and `workspaces:` policies to work with resources that you add to monitors so that Internet Monitor can create a traffic profile for your application. It contains some `iam:` policies to manage IAM roles.	October 23, 2024

Change	Description	Date
CloudWatchLambdaApplicationSignalsExecutionRolePolicy – New **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024	
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch updated the policy named **CloudWatchSyntheticsFullAccess**. The `lambda:ListTags` , `lambda:TagResource` , and `lambda:UntagResource` permissions were added so that when you apply or change tags on a canary, you can choose to have Synthetics also apply those same tags or changes to the Lambda function that the canary uses.	October 11, 2024

Change	Description	Date
CloudWatchApplicationSignalsReadOnlyAccess – New policy	CloudWatch created a new policy named **CloudWatchApplicationSignalsReadOnlyAccess**. This policy grants read only access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes `application-signals:` policies so that users can use read only actions and resources available in the CloudWatch console under Application Signals. It contains an `iam:` policy to manage IAM roles. It includes some `logs:` policies to manage log queries and filters. It includes `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `synthetics:` policies to retrive information about synthetics canaries. It includes `rum:` policies to manage RUM clients and jobs. It contains an `xray:` policy to obtain trace summaries.	June 7, 2024

Change	Description	Date
CloudWatchApplicationSignalsFullAccess – New policy	CloudWatch created a new policy named **CloudWatchApplicationSignalsFullAccess**. This policy grants full access to resources and actions available in the CloudWatch console for Application Signals. The scope of this policy includes `application-signals:` so that users can use Application Signals actions and resources. It includes some `cloudwatch:` policies to retrieve information on CloudWatch alarms and metrics. It includes some `logs:` policies to manage log queries. It includes some `synthetics:` policies to write and retrieve information on synthetics canaries. It includes `rum:` policies to manage RUM clients and jobs. It contains an `xray:` policy to obtain trace summaries. It includes some `cloudwatch:` policies to manage CloudWatch alarms. It contains some `iam:` policies to manage IAM roles. It includes some `sns:`	June 7, 2024

Change	Description	Date
	policies to manage Amazon Simple Notification Service notifications.	
CloudWatchFullAccessV2 – Update to an existing policy	CloudWatch updated the policy named **CloudWatchFullAccessV2**. The scope of the `CloudWatchFullAccessPermissions` policy was updated to add `application-signals:*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.	May 20, 2024

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch updated the policy named **CloudWatchReadOnlyAccess**. The scope of the `CloudWatchReadOnlyAccessPermissions` policy was updated to add `application-signals:BatchGet*`, `application-signals:List*`, and `application-signals:Get*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services. The scope of `CloudWatchReadOnlyGetRolePermissions` was updated to add the `iam:GetRole` action so that users can check if CloudWatch Application Signals is set up.	May 20, 2024

Change	Description	Date
CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy	CloudWatch updated the policy named **CloudWatchApplicationSignalsServiceRolePolicy**. The scoping of the `logs:StartQuery` and `logs:GetQueryResults` permissions was changed to add the `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` and `arn:aws:logs:*:*:log-group:/aws/application-signals/data:*` ARNs to enable Application Signals on more architectures.	April 18, 2024
CloudWatchApplicationSignalsServiceRolePolicy – Update to an existing policy	CloudWatch changed the scope of a permission in **CloudWatchApplicationSignalsServiceRolePolicy**. The scope of the `cloudwatch:GetMetricData` permission was changed to * so that Application Signals can retrieve metrics from sources in linked accounts.	April 08, 2024

Change	Description	Date
CloudWatchAgentServerPolicy – Update to an existing policy	CloudWatch added permissions to **CloudWatchAgentServerPolicy**. The `xray:PutTraceSegments` , `xray:PutTelemetryRecords` , `xray:GetSamplingRules` , `xray:GetSamplingTargets` , `xray:GetSamplingStatisticSummaries` and `logs:PutRetentionPolicy` permissions were added so that the CloudWatch agent can publish X-Ray traces and modify log group retention periods.	February 12, 2024

Change	Description	Date
CloudWatchAgentAdminPolicy – Update to an existing policy	CloudWatch added permissions to **CloudWatchAgentAdminPolicy**. The `xray:PutTraceSegments`, `xray:PutTelemetryRecords`, `xray:GetSamplingRules`, `xray:GetSamplingTargets`, `xray:GetSamplingStatisticSummaries` and `logs:PutRetentionPolicy` permissions were added so that the CloudWatch agent can publish X-Ray traces and modify log group retention periods.	February 12, 2024

Change	Description	Date
CloudWatchFullAccessV2 – Update to an existing policy	CloudWatch added permissions to **CloudWatchFullAccessV2**. Existing permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new permissions for CloudWatch Application Signals were added so that users with this policy can manage CloudWatch Application Signals. The permission to create the CloudWatch Application Signals service-linked role was added to allow CloudWatch Application Signals to discover telemetry data in logs, metrics, traces, and tags.	December 5, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. Existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM actions and new read-only permissions for CloudWatch Application Signals were added so that users with this policy can triage and dignose their service health issues as reported by CloudWatch Application Signals. The `cloudwatch:GenerateQuery` permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 5, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy.	CloudWatch added a permission to **CloudWatchReadOnlyAccess**. The `cloudwatch:GenerateQuery` permission was added, so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 01, 2023
CloudWatchApplicationSignalsServiceRolePolicy – New policy	CloudWatch added a new policy **CloudWatchApplicationSignalsServiceRolePolicy**. The **CloudWatchApplicationSignalsServiceRolePolicy** grants an upcoming feature permissions to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data.	November 9, 2023

Change	Description	Date
AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy – New policy	CloudWatch added a new policy **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy**. The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy** grants permission to CloudWatch to fetch Performance Insights metrics from databases on your behalf.	September 20, 2023
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added a permission to **CloudWatchReadOnlyAccess**. The `application-autoscaling:DescribeScalingPolicies` permission was added so that users with this policy can access information about Application Auto Scaling policies.	September 14, 2023

Change	Description	Date
CloudWatchFullAccessV2 – New policy	CloudWatch added a new policy **CloudWatchFullAccessV2**. The **CloudWatchFullAccessV2** grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling. For more information, see CloudWatchFullAccessV2.	August 1, 2023
AWSServiceRoleForInternetMonitor – Update to an existing policy	Amazon CloudWatch Internet Monitor added new permissions to monitor Network Load Balancer resources. The `elasticloadbalancing:DescribeLoadBalancers` and `ec2:DescribeNetworkInterfaces` permissions are required so that Internet Monitor can monitor customers' Network Load Balancer traffic by analyzing flow logs for NLB resources. For more information, see Using Internet Monitor.	July 15, 2023

Change	Description	Date
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `logs:StartLiveTail` and `logs:StopLiveTail` permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.	June 6, 2023
CloudWatchCrossAccountSharingConfiguration – New policy	CloudWatch added a new policy to enable you to manage CloudWatch cross-account observability links that share CloudWatch metrics. For more information, see CloudWatch cross-account observability.	November 27, 2022
OAMFullAccess – New policy	CloudWatch added a new policy to enable you to fully manage CloudWatch cross-account observability links and sinks. For more information, see CloudWatch cross-account observability.	November 27, 2022

Change	Description	Date
OAMReadOnlyAccess – New policy	CloudWatch added a new policy to enable you to view information about CloudWatch cross-account observability links and sinks. For more information, see CloudWatch cross-account observability.	November 27, 2022
CloudWatchFullAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchFullAccess**. The `oam:ListSinks` and `oam:ListAttachedLinks` permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022
CloudWatchReadOnlyAccess – Update to an existing policy	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `oam:ListSinks` and `oam:ListAttachedLinks` permissions were added so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability.	November 27, 2022

Change	Description	Date
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	CloudWatch RUM updated a condition key in **AmazonCloudWatchRUMServiceRolePolicy**. The `"Condition": { "StringEquals": { "cloudwatch:namespace": "AWS/RUM" } }` condition key was changed to the following so that CloudWatch RUM can send custom metrics to custom metric namespaces. `"Condition": { "StringLike": { "cloudwatch:namespace": [ "RUM/CustomMetrics/*", "AWS/RUM" ] } }`	February 2, 2023

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – Updated policy	CloudWatch added permissions the **AmazonCloudWatchRUMReadOnlyAccess** policy. The `rum:ListRumMetricsDestinations` and `rum:BatchGetRumMetricsDefinitions` permissions were added so that CloudWatch RUM can send extended metrics to CloudWatch.	October 27, 2022
AmazonCloudWatchRUMServiceRolePolicy – Update to an existing policy	CloudWatch RUM added permissions to **AmazonCloudWatchRUMServiceRolePolicy**. The `cloudwatch:PutMetricData` permission was added so that CloudWatch RUM can send extended metrics to CloudWatch.	October 26, 2022

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added permissions to **CloudWatchSyntheticsFullAccess**. The `lambda:DeleteFunction` and `lambda:DeleteLayerVersion` permissions were added so that CloudWatch Synthetics can delete related resources when a canary is deleted. The `iam:ListAttachedRolePolicies` was added so that customers can view the policies that are attached to a canary's IAM role.	May 6, 2022
AmazonCloudWatchRUMFullAccess – New policy	CloudWatch added a new policy to enable full management of CloudWatch RUM. CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.	November 29, 2021

Change	Description	Date
AmazonCloudWatchRUMReadOnlyAccess – New policy	CloudWatch added a new policy to enable read-only access to CloudWatch RUM. CloudWatch RUM allows you to perform real user monitoring of your web application. For more information, see CloudWatch RUM.	November 29, 2021
AWSServiceRoleForCloudWatchRUM – New managed policy	CloudWatch added a policy for a new service-linked role to allow CloudWatch RUM to pubish monitoring data to other relevant AWS services.	November 29, 2021

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added permissions to **CloudWatchSyntheticsFullAccess**, and also changed the scope of one permission. The `kms:ListAliases` permission was added so that users can list available AWS KMS keys that can be used to encrypt canary artifacts. The `kms:DescribeKey` permission was added so that users can see the details of keys that will be used to encrypt for canary artifacts. And the `kms:Decrypt` permission was added to enable users to decrypt canary artifacts. This decryption ability is limited to use on resources within Amazon S3 buckets. The `Resource` scope of the `s3:GetBucketLocation` permission was changed from * to `arn:aws:s3:::*` .	September 29, 2021

Change	Description	Date
CloudWatchSyntheticsFullAccess – Update to an existing policy	CloudWatch Synthetics added a permission to **CloudWatchSyntheticsFullAccess**. The `lambda:UpdateFunctionCode` permission was added so that users with this policy can change the runtime version of canaries.	July 20, 2021
AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy – New managed policy	CloudWatch added a new managed IAM policy to allow CloudWatch to create incidents in AWS Systems Manager Incident Manager.	May 10, 2021
CloudWatchAutomaticDashboardsAccess – Update to an existing policy	CloudWatch added a permission to the `CloudWatchAutomaticDashboardsAccess` managed policy. The `synthetics:DescribeCanariesLastRun` permission was added to this policy to enable cross-account dashboard users to see details about CloudWatch Synthetics canary runs.	April 20, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its AWS managed policies.	April 14, 2021

CloudWatchFullAccessV2

AWS recently added the **CloudWatchFullAccessV2** managed IAM policy. This policy grants full access to CloudWatch actions and resources and also more properly scopes the permissions granted for other services such as Amazon SNS and Amazon EC2 Auto Scaling. We recommend that you begin using this policy instead of using **CloudWatchFullAccess**. AWS plans to deprecate **CloudWatchFullAccess** in the near future.

It includes `application-signals:` permissions so that users can access all the functionality from the CloudWatch console under Application Signals. It includes some `autoscaling:Describe` permissions so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes some `sns` permissions so that users with this policy can retrieve create Amazon SNS topics and associate them with CloudWatch alarms. It includes IAM permissions so that users with this policy can view information about service-linked roles associated with CloudWatch. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability. It also includes CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals.

It includes Amazon OpenSearch Service permissions to support vended logs dashboards in CloudWatch Logs, which are created with Amazon OpenSearch Service analytics. It includes `resource-explorer-2:` policies so that users can use the console to view unstrumented services in their account.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have full access to CloudWatch Synthetics, AWS X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

It includes the `kms:DescribeKey` permission. CloudWatch Logs requires this permission from the caller to create a log group with a customer managed AWS KMS key or to associate a key with a log group. You also need it to encrypt the `aws/spans` log group with a customer managed key when you enable Transaction Search. You can use this permission only for requests that CloudWatch Logs makes on your behalf, and only for keys in your own account.

To see the full contents of the policy, see CloudWatchFullAccessV2 in the *AWS Managed Policy Reference Guide*.

CloudWatchFullAccess

The **CloudWatchFullAccess** policy is on the path to deprecation. We recommend that you stop using it, and use CloudWatchFullAccessV2 instead.

CloudWatchReadOnlyAccess

The **CloudWatchReadOnlyAccess** policy grants read-only access to CloudWatch and related observability features.

The policy includes some `logs:` permissions, so that users with this policy can use the console to view CloudWatch Logs information and CloudWatch Logs Insights queries. It includes `autoscaling:Describe*`, so that users with this policy can see the Auto Scaling actions that are associated with CloudWatch alarms. It includes the `application-signals:` permissions so that users can use Application Signals to monitor the health of their services. It includes `application-autoscaling:DescribeScalingPolicies`, so that users with this policy can access information about Application Auto Scaling policies. It includes `sns:Get*` and `sns:List*`, so that users with this policy can retrieve information about the Amazon SNS topics that receive notifications about CloudWatch alarms. It includes the `oam:ListSinks` and `oam:ListAttachedLinks` permissions, so that users with this policy can use the console to view data shared from source accounts in CloudWatch cross-account observability. It includes the `iam:GetRole` permissions so that users can check if CloudWatch Application Signals have been set up. It also includes CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals. It includes observability administration permissions for viewing telemetry rules, centralization configurations, and resource telemetry data across AWS Organizations. It includes the `cloudwatch:GenerateQuery` permission, so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt. It includes `resource-explorer-2:` policies so that users can use the console to view unstrumented services in their account.

It includes `rum`, `synthetics`, and `xray` permissions so that users can have read-only access to CloudWatch Synthetics, AWS X-Ray, and CloudWatch RUM, all of which are under the CloudWatch service.

To see the full contents of the policy, see CloudWatchReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchActionsEC2Access

The **CloudWatchActionsEC2Access** policy grants read-only access to CloudWatch alarms and metrics in addition to Amazon EC2 metadata. It also grants access to the Stop, Terminate, and Reboot API actions for EC2 instances.

To see the full contents of the policy, see CloudWatchActionsEC2Access in the *AWS Managed Policy Reference Guide*.

CloudWatch-CrossAccountAccess

The **CloudWatch-CrossAccountAccess** managed policy is used by the **CloudWatch-CrossAccountSharingRole** IAM role. This role and policy enable users of cross-account dashboards to view automatic dashboards in each account that is sharing dashboards.

To see the full contents of the policy, see CloudWatch-CrossAccountAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchAutomaticDashboardsAccess

The `CloudWatchAutomaticDashboardsAccess` managed policy grants access to CloudWatch for non-CloudWatch APIs, so that resources such as Lambda functions can be displayed on CloudWatch automatic dashboards.

To see the full contents of the policy, see CloudWatchAutomaticDashboardsAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchAgentServerPolicy

The **CloudWatchAgentServerPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances to allow the CloudWatch agent to read information from the instance and write it to CloudWatch.

To see the full contents of the policy, see CloudWatchAgentServerPolicy in the *AWS Managed Policy Reference Guide*.

CloudWatchAgentAdminPolicy

The **CloudWatchAgentAdminPolicy** policy can be used in IAM roles that are attached to Amazon EC2 instances. This policy allows the CloudWatch agent to read information from the instance and write it to CloudWatch, and also to write information to Parameter Store.

To see the full contents of the policy, see CloudWatchAgentAdminPolicy in the *AWS Managed Policy Reference Guide*.

CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy

You can't attach `CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role named **AWSServiceRoleForNetworkFlowMonitor_Topology**. Using these permissions, as well as internal meta data information gathering (for performance efficiencies), this service-linked role gathers meta data about resource network configurations, such as describing route tables and gateways, for resources that this service monitors network traffic for. This meta data enables Network Flow Monitor to generate topology snapshots of the resources. When there is network degradation, Network Flow Monitor uses the topologies to provide insights into the location of issues in the network and to help determine attribution for issues.

To view the permissions for this policy, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy in the *AWS Managed Policy Reference*.

For more information, see Service-linked roles for Network Flow Monitor.

> **Note**
>
> You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for CloudWatch actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

AWS managed (predefined) policies for CloudWatch cross-account observability

The policies in this section grant permissions related to CloudWatch cross-account observability. For more information, see CloudWatch cross-account observability.

CloudWatchCrossAccountSharingConfiguration

The **CloudWatchCrossAccountSharingConfiguration** policy grants access to create, manage, and view Observability Access Manager links for sharing CloudWatch resources between accounts. For more information, see CloudWatch cross-account observability.

To see the full contents of the policy, see CloudWatchCrossAccountSharingConfiguration in the *AWS Managed Policy Reference Guide*.

OAMFullAccess

The **OAMFullAccess** policy grants access to create, manage, and view Observability Access Manager sinks and links, which are used for CloudWatch cross-account observability.

The **OAMFullAccess** policy by itself does not permit you to share observability data across links. To create a link to share CloudWatch metrics, you also need either **CloudWatchFullAccess** or **CloudWatchCrossAccountSharingConfiguration**. To create a link to share CloudWatch Logs log groups, you also need either **CloudWatchLogsFullAccess** or **CloudWatchLogsCrossAccountSharingConfiguration**. To create a link to share X-Ray traces, you also need either **AWSXRayFullAccess** or **AWSXRayCrossAccountSharingConfiguration**.

For more information, see CloudWatch cross-account observability.

To see the full contents of the policy, see OAMFullAccess in the *AWS Managed Policy Reference Guide*.

OAMReadOnlyAccess

The **OAMReadOnlyAccess** policy grants read-only access to Observability Access Manager resources, which are used for CloudWatch cross-account observability. For more information, see CloudWatch cross-account observability.

To see the full contents of the policy, see OAMReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AWS managed (predefined) policies for CloudWatch investigations

The policies in this section grant permissions related to CloudWatch investigations. For more information, see CloudWatch investigations.

AIOpsConsoleAdminPolicy

The **AIOpsConsoleAdminPolicy** policy grants full access to all CloudWatch investigations actions and their required permissions through the AWS console. This policy also grants limited access to other service's APIs required for CloudWatch investigations functionality.

- The `aiops` permissions grant access to all CloudWatch investigations actions.

- The `organizations`, `sso`, `identitystore`, and `sts` permissions allow actions needed for IAM Identity Center management which help identity-aware sessions.
- The `ssm` permissions are required for SSM Ops Item integration with third-party issue management.
- The `iam` permissions are needed so that administrators can pass IAM roles to the `aiops` and `ssm.integrations` services, and the role is later used by the assistant to analyze AWS resources

> **Important**
>
> These permissions allow users with this policy to pass any IAM role to the `aiops` and `ssm.integrations` services.

- It allows APIs from services outside CloudWatch investigations, which are required for investigation feature functionality. These include actions to configure Amazon Q Developer in chat applications, AWS KMS, CloudTrail trails, and SSM third-party issue management.
- The `q` permissions enable integration with Amazon Q, allowing users to interact with and update CloudWatch investigations reports through Amazon Q's conversational interface.

To see the full contents of the policy, see AIOpsConsoleAdminPolicy in the *AWS Managed Policy Reference Guide*.

AIOpsOperatorAccess

The **AIOpsOperatorAccess** policy grants access to a limited set of CloudWatch investigations APIs which include creating, updating, and deleting investigations, investigation events, and investigation resources.

This policy only provides access to investigations. You should be sure that IAM principals with this policy also have permissions to read CloudWatch observability data such as metrics, SLOs, and CloudWatch Logs query results.

- The `aiops` permissions allow access to CloudWatch investigations APIs to create, update, and delete investigations.
- The `sso-directory`, `sso`, `identitystore`, and `sts` permissions allow actions needed for IAM Identity Center management which help identity-aware sessions.

- The `ssm` permissions are required for SSM Ops Item integration with third-party issue management.
- The `q` permissions enable integration with Amazon Q, allowing users to interact with and update CloudWatch investigations reports through Amazon Q's conversational interface.

To see the full contents of the policy, see AIOpsOperatorAccess in the *AWS Managed Policy Reference Guide*.

AIOpsReadOnlyAccess

The **AIOpsReadOnlyAccess** policy grants read-only permissions for CloudWatch investigations and other related services.

- The `aiops` permissions allow access to CloudWatch investigations APIs to get, list, and validate investigation groups.
- The `sso` permissions allow actions needed for IAM Identity Center management which help identity-aware sessions.
- The `ssm` permissions are required for SSM Ops Item integration with third-party issue management.

To see the full contents of the policy, see AIOpsReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AIOpsAssistantPolicy

The **AIOpsAssistantPolicy** policy is the default policy recommended by AWS to assign to the Amazon AI Operations (AIOps) role used by your investigation group to enable it to analyze your AWS resources during investigations of operational events. This policy is not intended for use by human users.

You can choose to have the policy assigned automatically when you create an investigation, or you can assign the policy manually to the role being used by the investigation. This policy is scoped based on the resources that CloudWatch investigations analyzes when performing investigations, and will be updated as more resources are supported. For a complete list of services that work with CloudWatch investigations see, AWS services where investigations are supported.

You can also choose to assign the general AWS **ReadOnlyAccess** to the assistant in addition to assigning it **AIOpsAssistantPolicy**. The reason to do this is that **ReadOnlyAccess** will be updated

more frequently by AWS with permissions for new AWS services and actions that are released. The **AIOpsAssistantPolicy** will also be updated for new actions, but not as frequently.

To see the full contents of the policy, see AIOpsAssistantPolicy in the *AWS Managed Policy Reference Guide*.

AIOpsAssistantIncidentReportPolicy

The **AIOpsAssistantIncidentReportPolicy** policy grants permissions required by CloudWatch investigations to generate incident reports from investigation data.

This policy is intended for use by CloudWatch investigations to enable automated incident report generation from investigation findings.

- The `aiops` permissions allow access to CloudWatch investigations APIs to read investigation data and events, create and update incident reports, and manage AI-derived facts that form the foundation of report generation.

To see the full contents of the policy, see AIOpsAssistantIncidentReportPolicy in the *AWS Managed Policy Reference Guide*.

AWS managed (predefined) policies for CloudWatch Application Signals

The policies in this section grant permissions related to CloudWatch Application Signals. For more information, see Application Signals.

CloudWatchApplicationSignalsReadOnlyAccess

AWS has added the **CloudWatchApplicationSignalsReadOnlyAccess** managed IAM policy. This policy grants read only access to actions and resources available to users in the CloudWatch console under Application Signals. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It includes an `iam:GetRole` policy to allow users to retrieve information about an IAM role. It includes `logs:` policies to start and stop queries, retrieve the configuration for a metruc filter, and obtain query results. It includes `cloudwatch:` policies so that users can obtain information about a CloudWatch alarm or metrics. It includes `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data, and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries. It includes `oam:` policies so that users can use the console to view data shared from source accounts in CloudWatch cross-account observability. It includes `resource-`

`explorer-2:` policies so that users can use the console to view unstrumented services in their account.

To see the full contents of the policy, see CloudWatchApplicationSignalsReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchApplicationSignalsFullAccess

AWS has added the **CloudWatchApplicationSignalsFullAccess** managed IAM policy. This policy grants access to all actions and resources available to users in the CloudWatch console. It includes `application-signals:` policies so that users can use CloudWatch Application signals to view, investigate and monitor the health of their services. It uses `cloudwatch:` policies to retrieve data from metrics and alarms. It uses `logs:` policies to manage queries and filters. It uses `synthetics:` policies so that users can retrieve information about synthetics canary runs. It includes `rum:` policies to run batch operations, retrieve data and update metrics definitions for RUM clients. It includes an `xray:` policy to retrieve trace summaries. It includes `arn:aws:cloudwatch:*:*:alarm:` policies so that users can retrieve information about a service level objective (SLO) alarm. It includes `iam:` policies to manage IAM roles. It uses `sns:` policies to create, list, and subscribe to an Amazon SNS topic. It includes `oam:` policies so that users can use the console to view data shared from source accounts in CloudWatch cross-account observability. It includes `resource-explorer-2:` policies so that users can use the console to view unstrumented services in their account

To see the full contents of the policy, see CloudWatchApplicationSignalsFullAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchLambdaApplicationSignalsExecutionRolePolicy

This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.

To see the full contents of the policy, see CloudWatchLambdaApplicationSignalsExecutionRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed (predefined) policies for CloudWatch Synthetics

The **CloudWatchSyntheticsFullAccess** and **CloudWatchSyntheticsReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch Synthetics. The following additional policies are also relevant:

- **AmazonS3ReadOnlyAccess** and **CloudWatchReadOnlyAccess** – These are necessary to read all Synthetics data in the CloudWatch console.
- **AWSLambdaReadOnlyAccess** – Required to view the source code used by canaries.
- **CloudWatchSyntheticsFullAccess** – Allows you to create canaries. Additionally, to create and delete canaries that have a new IAM role created for them, you need specific inline policy permissions.

> **Important**
>
> Granting a user the `iam:CreateRole`, `iam:DeleteRole`, `iam:CreatePolicy`, `iam:DeletePolicy`, `iam:AttachRolePolicy`, and `iam:DetachRolePolicy` permissions gives that user full administrative access to create, attach, and delete roles and policies that have ARNs that match `arn:aws:iam::*:role/service-role/CloudWatchSyntheticsRole*` and `arn:aws:iam::*:policy/service-role/CloudWatchSyntheticsPolicy*`. For example, a user with these permissions can create a policy that has full permissions for all resources, and attach that policy to any role that matches that ARN pattern. Be careful to whom you grant these permissions.

For information about attaching policies and granting permissions to users, see Changing Permissions for an IAM User and To embed an inline policy for a user or role.

CloudWatchSyntheticsFullAccess

To see the full contents of the policy, see CloudWatchSyntheticsFullAccess in the *AWS Managed Policy Reference Guide*.

CloudWatchSyntheticsReadOnlyAccess

To see the full contents of the policy, see CloudWatchSyntheticsReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AWS managed (predefined) policies for Amazon CloudWatch RUM

The **AmazonCloudWatchRUMFullAccess** and **AmazonCloudWatchRUMReadOnlyAccess** AWS managed policies are available for you to assign to users who will manage or use CloudWatch RUM.

AmazonCloudWatchRUMFullAccess

To see the full contents of the policy, see AmazonCloudWatchRUMFullAccess in the *AWS Managed Policy Reference Guide*.

AmazonCloudWatchRUMReadOnlyAccess

The **AmazonCloudWatchRUMReadOnlyAccess** allows read-only administrative access to CloudWatch RUM.

- The `synthetics` permission allows displaying associated Synthetics Canaries to the RUM app monitor
- The `cloudwatch` permission allows displaying associated CloudWatch metrics to the RUM app monitor
- The `cloudwatch alarms` permission allows displaying associated CloudWatch alarms to the RUM app monitor
- The `cloudwatch logs` permission allows displaying associated CloudWatch logs to the RUM app monitor
- The `x-ray` permission allows displaying associated X-Ray Trace segments to the RUM app monitor
- The `rum` permission allows displaying associated tags to the RUM app monitor

To see the full contents of the policy, see AmazonCloudWatchRUMReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AmazonCloudWatchRUMServiceRolePolicy

You can't attach **AmazonCloudWatchRUMServiceRolePolicy** to your IAM entities. This policy is attached to a service-linked role that allows CloudWatch RUM to publish monitoring data to other relevant AWS services. For more information about this service linked role, see Using service-linked roles for CloudWatch RUM.

To see the full contents of the policy, see AmazonCloudWatchRUMServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy for AWS Systems Manager Incident Manager

The **AWSCloudWatchAlarms_ActionSSMIncidentsServiceRolePolicy** policy is attached to a service-linked role that allows CloudWatch to start incidents in AWS Systems Manager Incident

Manager on your behalf. For more information, see Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions.

The policy has the following permission:

- ssm-incidents:StartIncident

Customer managed policy examples

In this section, you can find example user policies that grant permissions for various CloudWatch actions. These policies work when you are using the CloudWatch API, AWS SDKs, or the AWS CLI.

Examples

- Example 1: Allow user full access to CloudWatch
- Example 2: Allow read-only access to CloudWatch
- Example 3: Stop or terminate an Amazon EC2 instance

Example 1: Allow user full access to CloudWatch

To grant a user full access to CloudWatch, you can use grant them the **CloudWatchFullAccess** managed policy instead of creating a customer-managed policy. The contents of the **CloudWatchFullAccess** are listed in CloudWatchFullAccess.

Example 2: Allow read-only access to CloudWatch

The following policy allows a user read-only access to CloudWatch and view Amazon EC2 Auto Scaling actions, CloudWatch metrics, CloudWatch Logs data, and alarm-related Amazon SNS data.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*",
        "cloudwatch:List*",
```

```
        "logs:Get*",
        "logs:Describe*",
        "logs:StartQuery",
        "logs:StopQuery",
        "logs:TestMetricFilter",
        "logs:FilterLogEvents",
        "logs:StartLiveTail",
        "logs:StopLiveTail",
        "sns:Get*",
        "sns:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Example 3: Stop or terminate an Amazon EC2 instance

The following policy allows an CloudWatch alarm action to stop or terminate an EC2 instance. In the sample below, the GetMetricData, ListMetrics, and DescribeAlarms actions are optional. It is recommended that you include these actions to make sure that you have correctly stopped or terminated the instance.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:DescribeAlarms"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Using condition keys to limit access to CloudWatch

See the following topics to learn about the service-specific condition keys that can be used in the `Condition` element of an IAM policy. You can use these keys to further refine the conditions under which the policy statement applies.

- Using condition keys to limit access to CloudWatch namespaces
- Condition keys for Contributor Insights log group access
- Using condition keys to limit alarm actions
- Condition keys for CloudWatch Observability Admin

To see the global condition keys that are available to all services, see Available global condition keys.

Using condition keys to limit access to CloudWatch namespaces

Use IAM condition keys to limit users to publishing metrics only in the CloudWatch namespaces that you specify. This section provides examples that describe how to allow and exclude users from publishing metrics in a namespace.

Allowing publishing in one namespace only

The following policy limits the user to publishing metrics only in the namespace named `MyCustomNamespace`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Resource": "*",
        "Action": "cloudwatch:PutMetricData",
        "Condition": {
            "StringEquals": {
                "cloudwatch:namespace": "MyCustomNamespace"
            }
        }
    }
}
```

Excluding publishing from a namespace

The following policy allows the user to publish metrics in any namespace except for `CustomNamespace2`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        },
        {
            "Effect": "Deny",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "CustomNamespace2"
                }
            }
```

```
        }
    ]
}
```

Controlling OTLP ingest

The following policy allows the user to publish metrics using the OTLP API:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": "cloudwatch:PutMetricData"
        }
    ]
}
```

For disabling dual ingest, that is, only using PutMetricData and deny any OTLP ingest, you can use the following policy. It limits the user to publishing metrics using PutMetricData in the namespace `MyCustomNamespace` and at the same time implicitly denies any OTLP ingest due to the `StringEquals` condition:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                     "cloudwatch:namespace": "MyCustomNamespace"
                }
            }
         }
```

```
    ]
}
```

For enabling dual ingest, that is, to allow both PutMetricData and OTLP ingest, you can use the following policy. It limits the user to publishing metrics using PutMetricData in the namespace named `MyCustomNamespace` and at the same time allows OTLP ingest due to the `StringEqualsIfExists` condition:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                     "cloudwatch:namespace": "MyCustomNamespace"
                }
            }
         }
    ]
}
```

Condition keys for Contributor Insights log group access

To create a rule in Contributor Insights and see its results, a user must have the `cloudwatch:PutInsightRule` permission. By default, a user with this permission can create a rule that evaluates any log group in CloudWatch Logs and then view the results. Results can include contributor data from those log groups, which might contain sensitive information.

You can create IAM policies with condition keys to grant users permission to write Contributor Insights rules for specific log groups, while preventing access to data from other log groups.

For more information about the `Condition` element in IAM policies, see IAM JSON policy elements: Condition.

Understanding the Contributor Insights permissions model

Contributor Insights operations use the `cloudwatch:` IAM namespace. Log group operations use the `logs:` namespace. Contributor Insights does not require or evaluate `logs:` permissions when it processes log group data.

A principal with `cloudwatch:PutInsightRule` and `cloudwatch:GetInsightRuleReport` permissions can create rules that evaluate any log group and retrieve the results – even without any `logs:` permissions on those log groups.

> **Important**
>
> Aggregated results can contain sensitive information, such as log fields used as contributor keys. Grant `cloudwatch:PutInsightRule` and `cloudwatch:GetInsightRuleReport` permissions only to principals who need access to data across all referenced log groups.

For cross-account access, if a source account configured an AWS Organizations link that shares log group access, a principal in the monitoring account needs only `cloudwatch:PutInsightRule` to create rules that target source account log groups.

Restricting Contributor Insights access to specific log groups

Use the following condition keys to restrict which log groups a principal can specify when creating Contributor Insights rules:

- `cloudwatch:requestInsightRuleLogGroups` – Matches log group names specified in a rule
- `cloudwatch:requestInsightRuleLogGroupARNs` – Matches log group ARNs specified in a rule

> **Important**
>
> After a rule is created, any principal with `cloudwatch:GetInsightRuleReport` permission can retrieve its results, regardless of log group restrictions.

The following policy grants permission to create Contributor Insights rules for the log group named `AllowedLogGroup` and log groups with names that start with `AllowedWildCard`. It does not grant permission to create rules for any other log groups.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCertainLogGroups",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "AllowedLogGroup",
                        "AllowedWildcard*"
                    ]
                }
            }
        }
    ]
}
```

The following policy allows creating rules for any log group by default, but explicitly denies creating rules for the log group named `ExplicitlyDeniedLogGroup`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInsightRulesOnLogGroupsByDefault",
            "Effect": "Allow",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*"

        },
        {
            "Sid": "ExplicitDenySomeLogGroups",
            "Effect": "Deny",
```

```
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/*",
            "Condition": {
                "ForAllValues:StringEqualsIgnoreCase": {
                    "cloudwatch:requestInsightRuleLogGroups": [
                        "/test/alpine/ExplicitlyDeniedLogGroup"
                    ]
                }
            }
        }
    ]
}
```

The following policy denies creation of rules that target log groups with `/production/` in the ARN path.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyProductionLogGroupsByARN",
            "Effect": "Deny",
            "Action": "cloudwatch:PutInsightRule",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringLike": {
                    "cloudwatch:requestInsightRuleLogGroupARNs": [
                        "arn:aws:logs:*:*:log-group:*/production/*"
                    ]
                }
            }
        }
    ]
}
```

Note

- Condition keys match against the pattern strings as written in the rule definition, not against the resolved log groups.

- If rules specify log groups as full ARNs, a condition value such as `/production/*` does not match because the full ARN string does not start with that prefix. Use `*/production/*` as the wildcard prefix to match ARN-based log group references.

Security best practices for Contributor Insights

Follow these best practices to secure your Contributor Insights configuration.

- **Apply least privilege** – Grant `cloudwatch:PutInsightRule` and `cloudwatch:GetInsightRuleReport` only to principals who must analyze log group data
- **Use condition keys to restrict log groups** – Limit which log groups a principal can reference in Contributor Insights rules
- **Protect sensitive fields** – Use CloudWatch Logs data protection to mask sensitive log data before Contributor Insights processes it. For more information, see Protect sensitive log data with masking.
- **Consider contributor key sensitivity** – Contributor keys might expose values such as IP addresses or user identifiers in rule results
- **Review cross-account access** – Audit AWS Organizations link configurations to confirm that only intended monitoring accounts can create rules against source account log groups

Using condition keys to limit alarm actions

When CloudWatch alarms change state, they can perform different actions such as stopping and terminating EC2 instances and performing Systems Manager actions. These actions can be initiated when the alarm changes to any state, including ALARM, OK, or INSUFFICIENT_DATA.

Use the `cloudwatch:AlarmActions` condition key to allow a user to create alarms that can only perform the actions you specify when the alarm state changes. For example, you can allow a user to create alarms that can only perform actions which are not EC2 actions.

Allow a user to create alarms that can only send Amazon SNS notifications or perform Systems Manager actions

The following policy limits the user to creating alarms that can only send Amazon SNS notifications and perform Systems Manager actions. The user can't create alarms that perform EC2 actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAlarmsThatCanPerformOnlySNSandSSMActions",
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricAlarm",
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "cloudwatch:AlarmActions": [
                        "arn:aws:sns:*",
                        "arn:aws:ssm:*"
                    ]
                }
            }
        }
    ]
}
```

Condition keys for CloudWatch Observability Admin

You can use IAM policies to control access to Amazon CloudWatch Observability Admin resources and actions by using condition keys.

Observability Admin has the following condition keys:

Condition Key	Description	Type
CentralizationSourceRegions	ArrayOfString	Filters access by the source Regions that are passed in the request
CentralizationDestinationRegion	String	Filters access by the destination Region that is passed in the request

Condition Key	Description	Type
CentralizationBackupRegion	String	Filters access by the backup Region that is passed in the request

CentralizationSourceRegions

Filters access by the backup region specified for centralization rules.

- *Availability* – This key is available for the following resource types: organization-centralization-rule
- *Value type* – String

Example JSON policy with observabilityadmin:CentralizationBackupRegion

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
        "aws:RequestedRegion": "us-east-1"
        }
      }
    }
  ]
}
```

CentralizationDestinationRegion

Filters access by the destination region specified for centralization rules.

- *Availability* – This key is available for the following resource types: organization-centralization-rule
- *Value type* – String

Example JSON policy with observabilityadmin:CentralizationDestinationRegion

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
        "aws:RequestedRegion": "us-east-1"
        }
      }
    }
  ]
}
```

CentralizationBackupRegion

Filters access by the source regions specified for centralization rules.

- *Availability* – This key is available for the following resource types: organization-centralization-rule
- *Value type* – List of strings

Example JSON policy with observabilityadmin:CentralizationSourceRegions

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
        "aws:RequestedRegion": ["us-east-1", "us-east-1"]
        }
      }
    }
  ]
}
```

Condition keys for resource metrics configuration access

This topic explains how the `cloudwatch:ResourceArn` condition key works and how to use it to limit the resources that a principal can enable or disable detailed monitoring for.

The resource metrics configuration operations are not authorized against a CloudWatch resource type. These operations are `CreateResourceMetricsConfiguration`, `UpdateResourceMetricsConfiguration`, `GetResourceMetricsConfiguration`, and `DeleteResourceMetricsConfiguration`. Instead, they are authorized at the action level, and the AWS resource that the request targets is supplied to IAM in the `cloudwatch:ResourceArn` condition key. For links to the API reference for these operations, see Amazon CloudWatch permissions reference.

> **Important**
>
> Because these operations are authorized at the action level, specifying the target resource ARN in the `Resource` element of a policy statement does not restrict them. Use `"Resource": "*"` together with a `Condition` block on `cloudwatch:ResourceArn` to control which resources a principal can configure. A statement that omits the condition allows the principal to configure detailed monitoring for any resource in the account.

The value of `cloudwatch:ResourceArn` is the Amazon Resource Name (ARN) that the caller passes in the `ResourceArn` request parameter. Each of these operations targets exactly one resource per request.

For example, to allow a principal to manage detailed monitoring for only one resource, grant the resource metrics configuration actions on `"Resource": "*"` and add a `StringEquals` condition on `cloudwatch:ResourceArn` with the full ARN of that resource. A request that targets any other resource is denied.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
        "cloudwatch:CreateResourceMetricsConfiguration",
        "cloudwatch:UpdateResourceMetricsConfiguration",
        "cloudwatch:GetResourceMetricsConfiguration",
        "cloudwatch:DeleteResourceMetricsConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:ResourceArn": "arn:aws:service:us-
east-1:123456789012:resource-type/resource-id"
                }
            }
        }
    ]
}
```

`cloudwatch:ResourceArn` is evaluated as a string. Use `StringEquals` for an exact match, or `StringLike` with wildcards to match a set of resources. For example, you can use `arn:aws:service:*:123456789012:resource-type/*`. The ARN condition operators, such as `ArnEquals` and `ArnLike`, do not apply to this key.

The following policy uses `StringLike` with wildcards to allow the principal to manage detailed monitoring for any resource of one type in the account:

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
        "cloudwatch:CreateResourceMetricsConfiguration",
        "cloudwatch:UpdateResourceMetricsConfiguration",
        "cloudwatch:GetResourceMetricsConfiguration",
        "cloudwatch:DeleteResourceMetricsConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "cloudwatch:ResourceArn":
 "arn:aws:service:*:123456789012:resource-type/*"
                }
            }
        }
    ]
}
```

For more information about the `Condition` element in IAM policies, see IAM JSON policy elements: Condition.

Using service-linked roles for CloudWatch

Amazon CloudWatch uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CloudWatch. Service-linked roles are predefined by CloudWatch and include all the permissions that the service requires to call other AWS services on your behalf.

One service-linked role in CloudWatch makes setting up CloudWatch alarms that can terminate, stop, or reboot an Amazon EC2 instance without requiring you to manually add the necessary permissions. Another service-linked role enables a monitoring account to access CloudWatch data from other accounts that you specify, to build cross-account cross-Region dashboards.

CloudWatch defines the permissions of these service-linked roles, and unless defined otherwise, only CloudWatch can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This restriction protects your CloudWatch resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch alarms EC2 actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchEvents** – CloudWatch uses this service-linked role to perform Amazon EC2 alarm actions.

The AWSServiceRoleForCloudWatchEvents service-linked role trusts the CloudWatch Events service to assume the role. CloudWatch Events invokes the terminate, stop, or reboot instance actions when called upon by the alarm.

The AWSServiceRoleForCloudWatchEvents service-linked role permissions policy allows CloudWatch Events to complete the following actions on Amazon EC2 instances:

- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RecoverInstances`
- `ec2:DescribeInstanceRecoveryAttribute`
- `ec2:DescribeInstances`
- `ec2:DescribeInstanceStatus`

The **ServiceRoleForCloudWatchCrossAccountV2** service role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

Service-linked role permissions for CloudWatch telemetry config

CloudWatch observability admin creates and uses a service-linked role named **AWSServiceRoleForObservabilityAdmin** – CloudWatch uses this service-linked role to support resource and telemetry config discovery for AWS Organizations. When a management account

creates an organization-level rule, the service-linked role is automatically created in all member accounts and the delegated administrator account in the organization. However, when a delegated administrator account creates an organization-level rule, the service-linked role is automatically created in all member accounts but is *not* automatically created in the management account. If a delegated administrator account creates the organization-level rule, you must manually create the service-linked role in the management account.

The **AWSServiceRoleForObservabilityAdmin** service-linked role trusts Observability Admin to assume the role. Observability Admin manages AWS Config Service Linked Configuration Recorders and Service Linked Configuration Aggregator in your Organizations accounts.

The **AWSServiceRoleForObservabilityAdmin** service-linked role has a policy, called AWSObservabilityAdminServiceRolePolicy, attached and this policy grants permission to CloudWatch Observability Admin to complete the following actions:

- `organizations:ListAccounts`
- `organizations:ListAccountsForParent`
- `organizations:ListChildren`
- `organizations:ListParents`
- `organizations:DescribeOrganization`
- `organizations:DescribeOrganizationalUnit`
- `organizations:EnableAWSServiceAccess`
- `organizations:ListDelegatedAdministrators`
- `config:PutServiceLinkedConfigurationRecorder`
- `config:DeleteServiceLinkedConfigurationRecorder`
- `config:PutConfigurationAggregator`
- `config:DeleteConfigurationAggregator`
- `config:SelectAggregateResourceConfig`
- `iam:CreateServiceLinkedRole`
- `iam:PassRole`

The complete contents of the AWSObservabilityAdminServiceRolePolicy policy are as follows:

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "organizations:ListAccounts",
    "organizations:ListAccountsForParent",
    "organizations:ListChildren",
    "organizations:ListParents",
    "organizations:DescribeOrganization",
    "organizations:DescribeOrganizationalUnit"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "config:PutServiceLinkedConfigurationRecorder",
    "config:DeleteServiceLinkedConfigurationRecorder"
   ],
   "Resource": [
    "arn:aws:config:*:*:configuration-recorder/
AWSConfigurationRecorderForObservabilityAdmin/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "config:PutConfigurationAggregator",
    "config:DeleteConfigurationAggregator",
    "config:SelectAggregateResourceConfig"
   ],
   "Resource": [
    "arn:aws:config:*:*:config-aggregator/aws-service-config-aggregator/
observabilityadmin.amazonaws.com/*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
    "iam:CreateServiceLinkedRole"
   ],
   "Resource": [
    "arn:aws:iam::*:role/aws-service-role/config.amazonaws.com/
AWSServiceRoleForConfig"
   ],
   "Condition": {
    "StringEquals": {
     "iam:AWSServiceName": [
      "config.amazonaws.com"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": [
    "arn:aws:iam::*:role/aws-service-role/config.amazonaws.com/
AWSServiceRoleForConfig"
   ],
   "Condition": {
    "StringEquals": {
     "iam:PassedToService": [
      "config.amazonaws.com"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "organizations:EnableAWSServiceAccess"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "organizations:ServicePrincipal": [
      "config.amazonaws.com"
     ]
    }
   }
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "organizations:ListDelegatedAdministrators"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "organizations:ServicePrincipal": [
      "observabilityadmin.amazonaws.com",
      "config.amazonaws.com"
     ]
    }
   }
  }
 ]
}
```

Service-linked role permissions for CloudWatch telemetry enablement

The `AWSObservabilityAdminTelemetryEnablementServiceRolePolicy` grants permissions necessary to enable and manage telemetry configurations for AWS resources based on telemetry rules.

This policy grants permissions for:

- Basic telemetry operations including describing VPCs, flow logs, log groups. It also includes permissions to enable logging configuration for EKS cluster logging, WAF put logging configuration, enabling NLB logs, Route53 Resolver query logging, Amazon EC2 detailed monitoring, AWS Security Hub, Amazon Bedrock Agentcore Gateway, Amazon Bedrock Agentcore Memory, Amazon CloudFront distribution, S3 bucket server access logs, MSK Cluster, OpenTelemetry Enrichment, Amazon Bedrock Agentcore Workload Identity, Amazon Bedrock AgentCore Payment Manager, AWS X-Ray Transaction Search, and Amazon EKS Container Insights.
- Resource tagging operations with the `CloudWatchTelemetryRuleManaged` tag for tracking managed resources
- Log delivery configuration for services like AWS Bedrock and VPC Flow Logs
- Configuration recorder management for telemetry enablement tracking

The policy enforces security boundaries through conditions that:

- Restrict operations to resources within the same account using `aws:ResourceAccount`
- Require the `CloudWatchTelemetryRuleManaged` tag for resource modifications
- Limit configuration recorder access to those associated with telemetry enablement

The complete contents of the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy can be found here: AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.

Service-linked role permissions for CloudWatch Application Signals

CloudWatch Application Signals uses the service-linked role named **AWSServiceRoleForCloudWatchApplicationSignals** – CloudWatch uses this service-linked role to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.

The **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role trusts CloudWatch Application Signals to assume the role. Application Signals collects the logs, traces, metrics, and tags data from your account.

The **AWSServiceRoleForCloudWatchApplicationSignals** has an IAM policy attached, and this policy is named **CloudWatchApplicationSignalsServiceRolePolicy**. This policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant AWS services. It includes permissions for Application Signals to complete the following actions:

- `xray` – Retrieve X-Ray traces.
- `logs` – Retrieve the current CloudWatch logs information.
- `cloudwatch` – Retrieve the current CloudWatch metric information.
- `tags` – Retrieve the current tags.
- `application-signals` – Retrieve information on SLOs and their associated time exclusion windows.
- `autoscaling` – Retrieve application tags from Amazon EC2 Autoscaling group.
- `resource-explorer-2` – Retrieve current AWS resources information from AWS Resource Explorer.
- `cloudtrail` – Enable creation of service-linked channels for retrieving CloudTrail events.

The complete contents of **CloudWatchApplicationSignalsServiceRolePolicy** are as follows:

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "XRayPermission",
   "Effect": "Allow",
   "Action": [
    "xray:GetServiceGraph"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "CWLogsPermission",
   "Effect": "Allow",
   "Action": [
    "logs:StartQuery",
    "logs:GetQueryResults"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/appsignals/*:*",
    "arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "CWListMetricsPermission",
   "Effect": "Allow",
   "Action": [
```

```
    "cloudwatch:ListMetrics"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "CWGetMetricDataPermission",
   "Effect": "Allow",
   "Action": [
    "cloudwatch:GetMetricData"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Sid": "TagsPermission",
   "Effect": "Allow",
   "Action": [
    "tag:GetResources"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "ApplicationSignalsPermission",
   "Effect": "Allow",
   "Action": [
    "application-signals:ListServiceLevelObjectiveExclusionWindows",
       "application-signals:GetServiceLevelObjective"
   ],
   "Resource": [
```

```
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  },
  {
   "Sid": "EC2AutoScalingPermission",
   "Effect": "Allow",
   "Action": [
    "autoscaling:DescribeAutoScalingGroups"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

Service-linked role permissions for CloudWatch alarms Systems Manager OpsCenter actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchAlarms_ActionSSM** – CloudWatch uses this service-linked role to perform Systems Manager OpsCenter actions when a CloudWatch alarm goes into ALARM state.

The AWSServiceRoleForCloudWatchAlarms_ActionSSM service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager OpsCenter actions when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSM** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm:CreateOpsItem`

Service-linked role permissions for CloudWatch alarms Systems Manager Incident Manager actions

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** – CloudWatch uses this service-linked role to start Incident Manager incidents when a CloudWatch alarm goes into ALARM state.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role trusts the CloudWatch service to assume the role. CloudWatch alarms invoke the Systems Manager Incident Manager action when called upon by the alarm.

The **AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents** service-linked role permissions policy allows Systems Manager to complete the following actions:

- `ssm-incidents:StartIncident`

Service role permissions for CloudWatch cross-account cross-Region

CloudWatch uses the service role named **ServiceRoleForCloudWatchCrossAccountV2** – CloudWatch uses this role to access CloudWatch data in other AWS accounts that you specify. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the sharing account. It is the sharing role that provides access to data.

The **ServiceRoleForCloudWatchCrossAccountV2** service role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

The **ServiceRoleForCloudWatchCrossAccountV2** service role trusts the CloudWatch service to assume the role.

Service-linked role permissions for CloudWatch database Performance Insights

CloudWatch uses the service-linked role named **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** – CloudWatch uses this role to retrieve Performance Insights metrics for creating alarms and snapshotting.

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role has the `AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy` IAM policy attached. The contents of that policy are as follows:

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "pi:GetResourceMetrics"
   ],
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

The **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role trusts the CloudWatch service to assume the role.

Service-linked role permissions for CloudWatch Logs centralization

The **AWSObservabilityAdminLogsCentralizationServiceRolePolicy** attaches to appropriate IAM entity in your central management account, such as the CloudWatch service role –, to grant necessary permissions for setting up and managing centralized log collection. CloudWatch uses this role to access telemetry data from other AWS accounts that you specify to create CloudWatch log groups, log streams, and log events in your organization's monitoring account. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the monitoring account. This policy is attached automatically if you set up centralized log collection using the AWS Management Console. If you use the AWS CLI or API to configure log centralization you must attach this policy manually to the IAM role that will be used for observability administration tasks.

The **AWSObservabilityAdminLogsCentralizationServiceRolePolicy** service-linked role permissions policy allows CloudWatch to complete the following actions:

- `sts:AssumeRole`

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`
- `kms:Encrypt`
- `kms:Decrypt`
- `kms:GenerateDataKey`

The **AWSObservabilityAdminLogsCentralizationServiceRolePolicy** service-linked role trusts the `logs-centralization.observabilityadmin.amazonaws.com` service to assume the role.

Creating a service-linked role for CloudWatch

You do not need to manually create any of these service-linked roles. The first time you create an alarm in the AWS Management Console, the IAM CLI, or the IAM API, CloudWatch creates AWSServiceRoleForCloudWatchEvents and **AWSServiceRoleForCloudWatchAlarms_ActionSSM** for you.

The first time that you enable service and topology discovery, Application Signals creates **AWSServiceRoleForCloudWatchApplicationSignals** for you.

When you first enable an account to be a monitoring account for cross-account cross-Region functionality, CloudWatch creates **ServiceRoleForCloudWatchCrossAccountV2** for you.

When you first create an alarm that uses the `DB_PERF_INSIGHTS` metric math function, CloudWatch creates **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** for you.

For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Editing a service-linked role for CloudWatch

CloudWatch does not allow you to edit the **AWSServiceRoleForCloudWatchEvents**, **AWSServiceRoleForCloudWatchAlarms_ActionSSM**, **ServiceRoleForCloudWatchCrossAccountV2**, or **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** roles. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following commands:

   ```
   $ aws iam get-role --role-name role-name
   ```

   Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. To update a service-linked role's description, use the following command:

   ```
   $ aws iam update-role-description --role-name role-name --description description
   ```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

   GetRole
2. To update a role's description, use the following command:

   UpdateRoleDescription

Deleting a service-linked role for CloudWatch

If you no longer have alarms that automatically stop, terminate, or reboot EC2 instances, we recommend that you delete the AWSServiceRoleForCloudWatchEvents role.

If you not longer have alarms that perform Systems Manager OpsCenter actions, we recommend that you delete the AWSServiceRoleForCloudWatchAlarms_ActionSSM role.

If you delete all alarms that use the `DB_PERF_INSIGHTS` metric math function, we recommend that you delete the **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights** service-linked role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the AWSServiceRoleForCloudWatchEvents role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether CloudWatch is using the AWSServiceRoleForCloudWatchEvents role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call DeleteServiceLinkedRole. In the request, specify the role name that you want to delete.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.
2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

CloudWatch updates to AWS service-linked roles

View details about updates to AWS managed policies for CloudWatch since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudWatch Document history page.

Change	Description	Date
AWSObservabilityAdminTelemetryEnablementServiceRolePolicy – Update to service-linked role policy.	Updated information about the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the additional AWS resources based on telemetry rules.	April 30, 2026

Change	Description	Date
AWSObservabilityAdminTelemetryEnablementServiceRolePolicy – Update to service-linked role policy.	Updated information about the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the additional AWS resources based on telemetry rules.	March 31, 2026
AWSObservabilityAdminTelemetryEnablementServiceRolePolicy – Update to service-linked role policy.	Updated information about the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the additional AWS resources based on telemetry rules.	March 10, 2026
AWSObservabilityAdminTelemetryEnablementServiceRolePolicy – Update to service-linked role policy.	Updated information about the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the additional AWS resources based on telemetry rules.	December 2, 2025

Change	Description	Date
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	Updated the CloudWatchApplicationSignalsServiceRolePolicy to include `resource-explorer-2:Search` and `cloudtrail:CreateServiceLinkedChannel` to enable new Application Signals features.	November 12, 2025
AWSObservabilityAdminLogsCentralizationServiceRolePolicy – New service-linked role policy.	Added information about the AWSObservabilityAdminLogsCentralizationServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for AWS resources based on telemetry rules.	September 5, 2025
AWSObservabilityAdminTelemetryEnablementServiceRolePolicy – New service-linked role policy.	Added information about the AWSObservabilityAdminTelemetryEnablementServiceRolePolicy that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for AWS resources based on telemetry rules.	July 17, 2025

Change	Description	Date
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	Updated the CloudWatchApplicationSignalsServiceRolePolicy to exclude time windows from impacting the SLO attainment rate, error budget, and burn rate metrics. CloudWatch can retrieve exclusion windows on behalf of you.	March 13, 2025
AWSServiceRoleForObservabilityAdmin – New service-linked role	CloudWatch added this new service-linked role and corresponding managed policy, AWSObservabilityAdminServiceRolePolicy, to support resource and telemetry config discovery for AWS Organizations.	November 26, 2024
AWSServiceRoleForCloudWatchApplicationSignals – Update to permissions of service-linked role policy	CloudWatch add more log groups to the scope of the `logs:StartQuery` and `logs:GetQueryResults` permissions granted by this role.	April 24, 2024

Change	Description	Date
AWSServiceRoleForCloudWatchApplicationSignals – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch Application Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals.	November 9, 2023
AWSServiceRoleForCloudWatchMetrics_DbPerfInsights – New service-linked role	CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming and snapshotting. An IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 13, 2023
AWSServiceRoleForCloudWatchAlarms_ActionSSMIncidents – New service-linked role	CloudWatch added a new service-linked role to allow CloudWatch to create incidents in AWS Systems Manager Incident Manager.	April 26, 2021
CloudWatch started tracking changes	CloudWatch started tracking changes for its service-linked roles.	April 26, 2021

Using service-linked roles for CloudWatch RUM

CloudWatch RUM uses a AWS Identity and Access Management (IAM) service-linked role. A service-linked role is a unique type of IAM role that is linked directly to RUM. The service-linked role is predefined by RUM and includes all the permissions that the service requires to call other AWS services on your behalf.

RUM defines the permissions of the service-linked role, and unless defined otherwise, only RUM can assume the role. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the role only after first deleting its related resources. This restriction protects your RUM resources because you can't inadvertently remove permissions to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for RUM

RUM uses the service-linked role named **AWSServiceRoleForCloudWatchRUM** – this role allows RUM to send AWS X-Ray trace data into your account, for app monitors that you enable X-Ray tracing for.

The **AWSServiceRoleForCloudWatchRUM** service-linked role trusts the X-Ray service to assume the role. X-Ray sends the trace data to your account.

The **AWSServiceRoleForCloudWatchRUM** service-linked role has an IAM policy attached named **AmazonCloudWatchRUMServiceRolePolicy**. This policy grants permission to CloudWatch RUM to publish monitoring data to other relevant AWS services. It includes permissions that allow RUM to complete the following actions:

- `xray:PutTraceSegments`
- `cloudwatch:PutMetricData`

The complete contents of **AmazonCloudWatchRUMServiceRolePolicy** are as follows.

JSON

```
{
```

```
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "xray:PutTraceSegments"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringLike": {
     "cloudwatch:namespace": [
      "RUM/CustomMetrics/*",
      "AWS/RUM"
     ]
    }
   }
  }
 ]
}
```

Creating a service-linked role for RUM

You do not need to manually create the service-linked role for CloudWatch RUM. The first time that you create an app monitor with X-Ray tracing enabled, or update an app monitor to use X-Ray tracing, RUM creates **AWSServiceRoleForCloudWatchRUM** for you.

For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Editing a service-linked role for RUM

CloudWatch RUM does not allow you to edit the **AWSServiceRoleForCloudWatchRUM** role. After you create these roles, you cannot change their names because various entities might reference these roles. However, you can edit the description of these roles using IAM.

Editing a service-linked role description (IAM console)

You can use the IAM console to edit the description of a service-linked role.

To edit the description of a service-linked role (console)

1. In the navigation pane of the IAM console, choose **Roles**.
2. Choose the name of the role to modify.
3. To the far right of **Role description**, choose **Edit**.
4. Type a new description in the box, and choose **Save**.

Editing a service-linked role description (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to edit the description of a service-linked role.

To change the description of a service-linked role (AWS CLI)

1. (Optional) To view the current description for a role, use the following commands:

   ```
   $ aws iam get-role --role-name role-name
   ```

   Use the role name, not the ARN, to refer to roles with the AWS CLI commands. For example, if a role has the following ARN: `arn:aws:iam::123456789012:role/myrole`, you refer to the role as **`myrole`**.
2. To update a service-linked role's description, use the following command:

   ```
   $ aws iam update-role-description --role-name role-name --description description
   ```

Editing a service-linked role description (IAM API)

You can use the IAM API to edit the description of a service-linked role.

To change the description of a service-linked role (API)

1. (Optional) To view the current description for a role, use the following command:

   GetRole

2. To update a role's description, use the following command:

   UpdateRoleDescription

Deleting a service-linked role for RUM

If you no longer have app monitors with X-Ray enabled, we recommend that you delete the **AWSServiceRoleForCloudWatchRUM** role.

That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. Choose the name (not the check box) of the **AWSServiceRoleForCloudWatchRUM** role.
3. On the **Summary** page for the selected role, choose **Access Advisor** and review the recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether RUM is using the **AWSServiceRoleForCloudWatchRUM** role, try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

Deleting a service-linked role (IAM console)

You can use the IAM console to delete a service-linked role.

To delete a service-linked role (console)

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**. Select the check box next to the name of the role you want to delete, not the name or row itself.
3. For **Role actions**, choose **Delete role**.
4. In the confirmation dialog box, review the service last accessed data, which shows when each of the selected roles last accessed an AWS service. This helps you to confirm whether the role is currently active. To proceed, choose **Yes, Delete**.
5. Watch the IAM console notifications to monitor the progress of the service-linked role deletion. Because the IAM service-linked role deletion is asynchronous, the deletion task can succeed or fail after you submit the role for deletion. If the task fails, choose **View details** or **View Resources** from the notifications to learn why the deletion failed. If the deletion fails because there are resources in the service that are being used by the role, then the reason for the failure includes a list of resources.

Deleting a service-linked role (AWS CLI)

You can use IAM commands from the AWS Command Line Interface to delete a service-linked role.

To delete a service-linked role (AWS CLI)

1. Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `deletion-task-id` from the response to check the status of the deletion task. Type the following command to submit a service-linked role deletion request:

```
$ aws iam delete-service-linked-role --role-name service-linked-role-name
```

2. Type the following command to check the status of the deletion task:

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Deleting a service-linked role (IAM API)

You can use the IAM API to delete a service-linked role.

To delete a service-linked role (API)

1. To submit a deletion request for a service-linked role, call DeleteServiceLinkedRole. In the request, specify the role name that you want to delete.

   Because a service-linked role cannot be deleted if it is being used or has associated resources, you must submit a deletion request. That request can be denied if these conditions are not met. You must capture the `DeletionTaskId` from the response to check the status of the deletion task.
2. To check the status of the deletion, call GetServiceLinkedRoleDeletionStatus. In the request, specify the `DeletionTaskId`.

   The status of the deletion task can be `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. If the deletion fails, the call returns the reason that it failed so that you can troubleshoot.

Supported Regions for CloudWatch RUM service-linked roles

CloudWatch RUM supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see CloudWatch RUM service endpoints.

Using service-linked roles for CloudWatch Application Insights

CloudWatch Application Insights uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CloudWatch Application Insights. Service-linked roles are predefined by CloudWatch Application Insights and include all of the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CloudWatch Application Insights easier because you don't have to manually add the necessary permissions. CloudWatch Application Insights defines the permissions of its service-linked roles, and unless defined otherwise, only CloudWatch Application Insights can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudWatch Application Insights

CloudWatch Application Insights uses the service-linked role named **AWSServiceRoleForApplicationInsights**. Application Insights uses this role to perform operations such as analyzing the resource groups of the customer, creating CloudFormation stacks to create alarms on metrics, and configuring the CloudWatch Agent on EC2 instances. This service-linked role has an IAM policy attached to it named `CloudwatchApplicationInsightsServiceLinkedRolePolicy`. For updates to this policy, see Application Insights updates to AWS managed policies.

The role permissions policy allows CloudWatch Application Insights to complete the following actions on resources.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatch",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarmHistory",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:PutAnomalyDetector",
        "cloudwatch:DeleteAnomalyDetector",
        "cloudwatch:DescribeAnomalyDetectors"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",
```

```
        "logs:DescribeLogStreams",
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridge",
      "Effect": "Allow",
      "Action": [
        "events:DescribeRule"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudFormation",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:CreateStack",
        "cloudFormation:UpdateStack",
        "cloudFormation:DeleteStack",
        "cloudFormation:DescribeStackResources",
        "cloudFormation:UpdateTerminationProtection"
      ],
      "Resource": [
        "arn:aws:cloudformation:*:*:stack/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "CloudFormationStacks",
      "Effect": "Allow",
      "Action": [
        "cloudFormation:DescribeStacks",
        "cloudFormation:ListStackResources",
        "cloudFormation:ListStacks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
```

```
      "Sid": "Tag",
      "Effect": "Allow",
      "Action": [
        "tag:GetResources"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ResourceGroups",
      "Effect": "Allow",
      "Action": [
        "resource-groups:ListGroupResources",
        "resource-groups:GetGroupQuery",
        "resource-groups:GetGroup"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationInsightsResourceGroup",
      "Effect": "Allow",
      "Action": [
        "resource-groups:CreateGroup",
        "resource-groups:DeleteGroup"
      ],
      "Resource": [
        "arn:aws:resource-groups:*:*:group/ApplicationInsights-*"
      ]
    },
    {
      "Sid": "ElasticLoadBalancing",
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth"
      ],
      "Resource": [
        "*"
      ]
    },
```

```
    {
      "Sid": "AutoScaling",
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMParameter",
      "Effect": "Allow",
      "Action": [
        "ssm:PutParameter",
        "ssm:DeleteParameter",
        "ssm:AddTagsToResource",
        "ssm:RemoveTagsFromResource",
        "ssm:GetParameters"
      ],
      "Resource": "arn:aws:ssm:*:*:parameter/AmazonCloudWatch-
ApplicationInsights-*"
    },
    {
      "Sid": "SSMAssociation",
      "Effect": "Allow",
      "Action": [
        "ssm:CreateAssociation",
        "ssm:UpdateAssociation",
        "ssm:DeleteAssociation",
        "ssm:DescribeAssociation"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:association/*",
        "arn:aws:ssm:*:*:managed-instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-
ApplicationInsightsCloudwatchAgentInstallAndConfigure",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
      "Sid": "SSMOpsItem",
```

```
      "Effect": "Allow",
      "Action": [
        "ssm:GetOpsItem",
        "ssm:CreateOpsItem",
        "ssm:DescribeOpsItems",
        "ssm:UpdateOpsItem",
        "ssm:DescribeInstanceInformation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMTags",
      "Effect": "Allow",
      "Action": [
        "ssm:AddTagsToResource"
      ],
      "Resource": "arn:aws:ssm:*:*:opsitem/*"
    },
    {
      "Sid": "SSMGetCommandInvocation",
      "Effect": "Allow",
      "Action": [
        "ssm:ListCommandInvocations",
        "ssm:GetCommandInvocation"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SSMSendCommand",
      "Effect": "Allow",
      "Action": "ssm:SendCommand",
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ssm:*:*:document/AWSEC2-CheckPerformanceCounterSets",
        "arn:aws:ssm:*:*:document/AWS-ConfigureAWSPackage",
        "arn:aws:ssm:*:*:document/AWSEC2-DetectWorkload",
        "arn:aws:ssm:*:*:document/AmazonCloudWatch-ManageAgent"
      ]
    },
    {
```

```
      "Sid": "EC2",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeNatGateways"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "RDS",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Lambda",
      "Effect": "Allow",
      "Action": [
        "lambda:ListFunctions",
        "lambda:GetFunctionConfiguration",
        "lambda:ListEventSourceMappings"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EventBridgeManagedRule",
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
```

```
        "events:DeleteRule"
      ],
      "Resource": [
        "arn:aws:events:*:*:rule/AmazonCloudWatch-ApplicationInsights-*"
      ]
    },
    {
      "Sid": "XRay",
      "Effect": "Allow",
      "Action": [
        "xray:GetServiceGraph",
        "xray:GetTraceSummaries",
        "xray:GetTimeSeriesServiceStatistics",
        "xray:GetTraceGraph"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "DynamoDB",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ListTables",
        "dynamodb:DescribeTable",
        "dynamodb:DescribeContributorInsights",
        "dynamodb:DescribeTimeToLive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ApplicationAutoscaling",
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalableTargets"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3",
```

```
      "Effect": "Allow",
      "Action": [
        "s3:ListAllMyBuckets",
        "s3:GetMetricsConfiguration",
        "s3:GetReplicationConfiguration"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "States",
      "Effect": "Allow",
      "Action": [
        "states:ListStateMachines",
        "states:DescribeExecution",
        "states:DescribeStateMachine",
        "states:GetExecutionHistory"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "APIGateway",
      "Effect": "Allow",
      "Action": [
        "apigateway:GET"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECS",
      "Effect": "Allow",
      "Action": [
        "ecs:DescribeClusters",
        "ecs:DescribeContainerInstances",
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:DescribeTaskSets",
        "ecs:ListClusters",
```

```
        "ecs:ListContainerInstances",
        "ecs:ListServices",
        "ecs:ListTasks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "ECSCluster",
      "Effect": "Allow",
      "Action": [
        "ecs:UpdateClusterSettings"
      ],
      "Resource": [
        "arn:aws:ecs:*:*:cluster/*"
      ]
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:DescribeFargateProfile",
        "eks:DescribeNodegroup",
        "eks:ListClusters",
        "eks:ListFargateProfiles",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "fsx:DescribeVolumes"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SNS",
      "Effect": "Allow",
      "Action": [
        "sns:GetSubscriptionAttributes",
        "sns:GetTopicAttributes",
        "sns:GetSMSAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTopics"
```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "SQS",
      "Effect": "Allow",
      "Action": [
        "sqs:ListQueues"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsDeleteSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:DeleteSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*"
      ]
    },
    {
      "Sid": "CloudWatchLogsCreateSubscriptionFilter",
      "Effect": "Allow",
      "Action": [
        "logs:PutSubscriptionFilter"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:*",
        "arn:aws:logs:*:*:destination:AmazonCloudWatch-ApplicationInsights-
LogIngestionDestination*"
      ]
    },
    {
      "Sid": "EFS",
      "Effect": "Allow",
      "Action": [
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": [
        "*"
      ]
```

```
    },
    {
      "Sid": "Route53",
      "Effect": "Allow",
      "Action": [
        "route53:GetHostedZone",
        "route53:GetHealthCheck",
        "route53:ListHostedZones",
        "route53:ListHealthChecks",
        "route53:ListQueryLoggingConfigs"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Route53Resolver",
      "Effect": "Allow",
      "Action": [
        "route53resolver:ListFirewallRuleGroupAssociations",
        "route53resolver:GetFirewallRuleGroup",
        "route53resolver:ListFirewallRuleGroups",
        "route53resolver:ListResolverEndpoints",
        "route53resolver:GetResolverQueryLogConfig",
        "route53resolver:ListResolverQueryLogConfigs",
        "route53resolver:ListResolverQueryLogConfigAssociations",
        "route53resolver:GetResolverEndpoint",
        "route53resolver:GetFirewallRuleGroupAssociation"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a service-linked role for CloudWatch Application Insights

You don't need to manually create a service-linked role. When you create a new Application Insights application in the AWS Management Console, CloudWatch Application Insights creates the service-linked role for you.

If you delete this service-linked role, and then want to create it again, you can use the same process to recreate the role in your account. When you create a new Application Insights application, CloudWatch Application Insights creates the service-linked role for you again.

Editing a service-linked role for CloudWatch Application Insights

CloudWatch Application Insights does not allow you to edit the AWSServiceRoleForApplicationInsights service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for CloudWatch Application Insights

If you no longer need to use a feature or service that requires a service-linked role, the service recommend that you delete that role. That way you avoid having an unused entity that is not actively monitored or maintained. However, you must delete all applications in Application Insights before you can manually delete the role.

> **Note**
>
> If the CloudWatch Application Insights service is using the role when you try to delete the resources, the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CloudWatch Application Insights resources used by the AWSServiceRoleForApplicationInsights

- Delete all of your CloudWatch Application Insights applications. For more information, see "Deleting Your Application(s)" in the CloudWatch Application Insights User Guide.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForApplicationInsights service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for CloudWatch Application Insights service-linked roles

CloudWatch Application Insights supports using service-linked roles in all of the AWS Regions where the service is available. For more information, see CloudWatch Application Insights Regions and Endpoints.

AWS managed policies for Amazon CloudWatch Application Insights

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: CloudWatchApplicationInsightsFullAccess

You can attach the `CloudWatchApplicationInsightsFullAccess` policy to your IAM identities.

This policy grants administrative permissions that allow full access to Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows full access to Application Insights functionality.
- `iam` – Allows Application Insights to create the service-linked role, AWSServiceRoleForApplicationInsights. This is required so that Application Insights can perform operations such as analyze the resource groups of a customer, create CloudFormation stacks to create alarms on metrics, and configure the CloudWatch Agent on EC2 instances. For more information, see Using service-linked roles for CloudWatch Application Insights.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "applicationinsights:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "sqs:ListQueues",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "autoscaling:DescribeAutoScalingGroups",
```

```
        "lambda:ListFunctions",
        "dynamodb:ListTables",
        "s3:ListAllMyBuckets",
        "sns:ListTopics",
        "states:ListStateMachines",
        "apigateway:GET",
        "ecs:ListClusters",
        "ecs:DescribeTaskDefinition",
        "ecs:ListServices",
        "ecs:ListTasks",
        "eks:ListClusters",
        "eks:ListNodegroups",
        "fsx:DescribeFileSystems",
        "logs:DescribeLogGroups",
        "elasticfilesystem:DescribeFileSystems"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/application-insights.amazonaws.com/
AWSServiceRoleForApplicationInsights"
      ],
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "application-insights.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS managed policy: CloudWatchApplicationInsightsReadOnlyAccess

You can attach the `CloudWatchApplicationInsightsReadOnlyAccess` policy to your IAM identities.

This policy grants administrative permissions that allow read-only access to all Application Insights functionality.

Permissions details

This policy includes the following permissions.

- `applicationinsights` – Allows read-only access to Application Insights functionality.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "applicationinsights:Describe*",
                "applicationinsights:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: CloudwatchApplicationInsightsServiceLinkedRolePolicy

You can't attach CloudwatchApplicationInsightsServiceLinkedRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows Application Insights to monitor customer resources. For more information, see Using service-linked roles for CloudWatch Application Insights.

Application Insights updates to AWS managed policies

View details about updates to AWS managed policies for Application Insights since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Application Insights Document history page.

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added a new permission. The policy change allows Amazon CloudWatch Application Insights to enable and disable termination protection on CloudFormation stacks to manage SSM resources used to install and configure CloudWatch agents.	July 25, 2024
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to list CloudFormation stacks. These permissions are required for Amazon CloudWatch Application Insights to analyze and monitor AWS resources nested in the CloudFormation stack.	April 24, 2023
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to get list of Amazon VPC and Route 53 resources. These permissions are required for Amazon CloudWatch Application	January 23, 2023

Change	Description	Date
	Insights to automatically set up best practice network monitoring with Amazon CloudWatch.	
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to get SSM command invocation results. These permissions are required for Amazon CloudWatch Application Insights to automatically detect and monitor workloads running on Amazon EC2 instances.	December 19, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe Amazon VPC and Route 53 resources. These permissions are required for Amazon CloudWatch Application Insights to read customer Amazon VPC and Route 53 resource configurations, and to help customers automatic ally set up best practice network monitoring with Amazon CloudWatch.	December 19, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe EFS resources. These permissions are required for Amazon CloudWatch Application Insights to read Amazon EFS customer resource configurations, and to help customers automatically set up best practices for EFS monitoring with CloudWatch.	October 3, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe the EFS file system. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	October 3, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to retrieve information about FSx resources. These permissions are required for Amazon CloudWatch Application Insights to monitor workloads by retrieving sufficient information about the underlying FSx volumes.	September 12, 2022
AWS managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	Application Insights added a new permission to describe log groups. This permissions is required for Amazon CloudWatch Application Insights to make sure that the correct permissions for monitoring log groups are in an account when creating a new application.	January 24, 2022

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to create and delete CloudWatch Log Subscription Filters. These permissions are required for Amazon CloudWatch Application Insights to create Subscription Filters to help log monitoring of resources within configured applications.	January 24, 2022
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe target groups and target health for Elastic Load Balancers. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	November 4, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to run the `AmazonCloudWatch-ManageAgent` SSM document on Amazon EC2 instances. This permissions is required for Amazon CloudWatch Application Insights to clean up CloudWatch agent configuration files created by Application Insights.	September 30, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to support account-based application monitoring to onboard and monitor all supported resources in your account. These permissions are required for Amazon CloudWatch Application Insights to query, tag resources, and create groups for these resources. Application Insights added new permissions to support monitoring of SNS topics. These permissions are required for Amazon CloudWatch Application Insights to gather metadata from SNS resources to configure monitoring for SNS topics.	September 15, 2021

Change	Description	Date
AWS managed policy: CloudWatchApplicationInsightsFullAccess – Update to an existing policy	Application Insights added new permissions to describe and list supported resources. These permissions are required for Amazon CloudWatch Application Insights to create account-based applications by querying all of the supported resources in an account.	September 15, 2021
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe FSx resources. These permissions are required for Amazon CloudWatch Application Insights to read customer FSx resource configurations, and to help customers automatically set up best practice FSx monitoring with CloudWatch.	August 31, 2021

Change	Description	Date
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to describe and list ECS and EKS service resources. This permission is required for Amazon CloudWatch Application Insights to read customer container resources configuration, and to help customers automatically set up best practice container monitoring with CloudWatch.	May 18, 2021
CloudwatchApplicationInsightsServiceLinkedRolePolicy – Update to an existing policy	Application Insights added new permissions to allow OpsCenter to tag OpsItems using the `ssm:AddTagsToResource` action on resources with the `opsitem` resource type. This permission is required by OpsCenter. Amazon CloudWatch Application Insights creates OpsItems so that the customer can resolve problems using AWS SSM OpsCenter.	April 13, 2021
Application Insights started tracking changes	Application Insights started tracking changes for its AWS managed policies.	April 13, 2021

Amazon CloudWatch permissions reference

The following table lists each CloudWatch API operation and the corresponding actions for which you can grant permissions to perform the action. You specify the actions in the policy's `Action` field, and you specify a wildcard character (*) as the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your CloudWatch policies to express conditions. For a complete list of AWS-wide keys, see AWS Global and IAM Condition Context Keys in the *IAM User Guide*.

> **Note**
>
> To specify an action, use the `cloudwatch:` prefix followed by the API operation name. For example: `cloudwatch:GetMetricData`, `cloudwatch:ListMetrics`, or `cloudwatch:*` (for all CloudWatch actions).

Topics

- CloudWatch API operations and required permissions for actions
- CloudWatch Application Signals API operations and required permissions for actions
- CloudWatch Contributor Insights API operations and required permissions for actions
- CloudWatch Events API operations and required permissions for actions
- CloudWatch Logs API operations and required permissions for actions
- Amazon EC2 API operations and required permissions for actions
- Amazon EC2 Auto Scaling API operations and required permissions for actions

CloudWatch API operations and required permissions for actions

CloudWatch API operations	Required permissions (API actions)
CreateResourceMetricsConfiguration	`cloudwatch:CreateResourceMetricsConfiguration` Required to create a resource metrics configuration that enables detailed monitoring for a resource.

CloudWatch API operations	Required permissions (API actions)
	To restrict which resources a principal can target, use the `cloudwatch:ResourceArn` condition key. See Condition keys for resource metrics configuration access.
DeleteAlarms	`cloudwatch:DeleteAlarms` Required to delete an alarm.
DeleteDashboards	`cloudwatch:DeleteDashboards` Required to delete a dashboard.
DeleteMetricStream	`cloudwatch:DeleteMetricStream` Required to delete a metric stream.
DeleteResourceMetricsConfiguration	`cloudwatch:DeleteResourceMetricsConfiguration` Required to delete a resource metrics configuration and disable detailed monitoring for a resource. To restrict which resources a principal can target, use the `cloudwatch:ResourceArn` condition key. See Condition keys for resource metrics configuration access.

CloudWatch API operations	Required permissions (API actions)
DescribeAlarmHistory	`cloudwatch:DescribeAlarmHistory` Required to view alarm history. To retrieve information about composite alarms, your `cloudwatch:DescribeAlarmHistory` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:DescribeAlarmHistory` permission has a narrower scope.
DescribeAlarms	`cloudwatch:DescribeAlarms` Required to retrieve information about alarms. To retrieve information about composite alarms, your `cloudwatch:DescribeAlarms` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:DescribeAlarms` permission has a narrower scope.
DescribeAlarmsForMetric	`cloudwatch:DescribeAlarmsForMetric` Required to view alarms for a metric.
DisableAlarmActions	`cloudwatch:DisableAlarmActions` Required to disable an alarm action.
EnableAlarmActions	`cloudwatch:EnableAlarmActions` Required to enable an alarm action.

CloudWatch API operations	Required permissions (API actions)
GetDashboard	`cloudwatch:GetDashboard` Required to display data about existing dashboards.
GetMetricData	`cloudwatch:GetMetricData` Required to graph metric data in the CloudWatch console, to retrieve large batches of metric data, and perform metric math on that data.
GetMetricStatistics	`cloudwatch:GetMetricStatistics` Required to view graphs in other parts of the CloudWatch console and in dashboard widgets.
GetMetricStream	`cloudwatch:GetMetricStream` Required to view information about a metric stream.
GetMetricWidgetImage	`cloudwatch:GetMetricWidgetImage` Required to retrieve a snapshot graph of one or more CloudWatch metrics as a bitmap image.
GetOTelEnrichment	`cloudwatch:GetOTelEnrichment` Required to retrieve the status of OpenTelemetry enrichment for vended metrics.

CloudWatch API operations	Required permissions (API actions)
GetResourceMetricsConfiguration	`cloudwatch:GetResourceMetricsConfiguration` Required to retrieve the resource metrics configuration for a resource. To restrict which resources a principal can target, use the `cloudwatch:ResourceArn` condition key. See Condition keys for resource metrics configuration access.
ListDashboards	`cloudwatch:ListDashboards` Required to view the list of CloudWatch dashboards in your account.
ListEntitiesForMetric (CloudWatch console-only permission)	`cloudwatch:ListEntitiesForMetric` Required to find the entities associated with a metric. Required to explore related telemetry within the CloudWatch console.
ListMetrics	`cloudwatch:ListMetrics` Required to view or search metric names within the CloudWatch console and in the CLI. Required to select metrics on dashboard widgets.
ListMetricStreams	`cloudwatch:ListMetricStreams` Required to view or search the list of metric streams in the account.

CloudWatch API operations	Required permissions (API actions)
ListTagsForResource	`cloudwatch:ListTagsForResource` Required to list the tags associated with a CloudWatch resource.
PutCompositeAlarm	`cloudwatch:PutCompositeAlarm` Required to create a composite alarm. To create a composite alarm, your `cloudwatch:PutCompositeAlarm` permission must have a * scope. You can't return information about composite alarms if your `cloudwatch:PutCompositeAlarm` permission has a narrower scope.
PutDashboard	`cloudwatch:PutDashboard` Required to create a dashboard or update an existing dashboard.
PutMetricAlarm	`cloudwatch:PutMetricAlarm` Required to create or update an alarm.
PutMetricData	`cloudwatch:PutMetricData` Required to create metrics.
PutMetricStream	`cloudwatch:PutMetricStream` Required to create a metric stream.

CloudWatch API operations	Required permissions (API actions)
SetAlarmState	`cloudwatch:SetAlarmState` Required to manually set an alarm's state.
StartMetricStreams	`cloudwatch:StartMetricStreams` Required to start the flow of metrics in a metric stream.
StartOTelEnrichment	`cloudwatch:StartOTelEnrichment` Required to enable OpenTelemetry enrichment for vended metrics, making them queryable via PromQL.
StopMetricStreams	`cloudwatch:StopMetricStreams` Required to temporarily stop the flow of metrics in a metric stream.
StopOTelEnrichment	`cloudwatch:StopOTelEnrichment` Required to disable OpenTelemetry enrichment for vended metrics.
TagResource	`cloudwatch:TagResource` Required to add or update tags on CloudWatch resources such as alarms and Contributor Insights rules.

CloudWatch API operations	Required permissions (API actions)
UntagResource	`cloudwatch:UntagResource` Required to remove tags from CloudWatch resources .
UpdateResourceMetricsConfiguration	`cloudwatch:UpdateResourceMetricsConfiguration` Required to update a resource metrics configuration, including the metric selections for a resource. To restrict which resources a principal can target, use the `cloudwatch:ResourceArn` condition key. See Condition keys for resource metrics configuration access.

CloudWatch Application Signals API operations and required permissions for actions

CloudWatch Application Signals API operations	Required permissions (API actions)
BatchGetServiceLevelObjectiveBudgetReport	`application-signals:BatchGetServiceLevelObjectiveBudgetReport` Required to retrieve service level objective budget reports.
CreateServiceLevelObjective	`application-signals:CreateServiceLevelObjective`

CloudWatch Application Signals API operations	Required permissions (API actions)
	Required to create a service level objective (SLO).
DeleteServiceLevelObjective	`application-signals:DeleteServiceLevelObjective` Required to delete a service level objective (SLO).
GetService	`application-signals:GetService` Required to retrieve information about a service discovered by Application Signals.
GetServiceLevelObjective	`application-signals:GetServiceLevelObjective` Required to retrieve information about a service level objective (SLO).
ListObservedEntities	`application-signals:ListObservedEntities` Grants permission to list entities that are associated with other entities.
ListServiceDependencies	`application-signals:ListServiceDependencies` Required to retrieve a list of service dependencies of a service that you specify. This service and the dependencies were discovered by Application Signals.

CloudWatch Application Signals API operations	Required permissions (API actions)
ListServiceDependents	`application-signals:ListServiceDependents` Required to retrieve a list of dependents that invoked a service that you specify. This service and the dependents were discovered by Application Signals.
ListServiceLevelObjectives	`application-signals:ListServiceLevelObjectives` Required to retrieve a list of service level objectives (SLOs) in the account.
ListServiceOperations	`application-signals:ListServiceOperations` Required to retrieve a list of service operations of a service that you specify. This service and the dependencies were discovered by Application Signals.
ListServices	`application-signals:ListServices` Required to retrieve a list of services discovered by Application Signals.
ListTagsForResource	`application-signals:ListTagsForResource` Required to retrieve a list of the tags associated with a resource.

CloudWatch Application Signals API operations	Required permissions (API actions)
StartDiscovery	`application-signals:StartDiscovery` Required to be able to enable Application Signals in the account and create the required service-linked role.
TagResource	`application-signals:TagResource` Required to be able to add tags to resources.
UntagResource	`application-signals:UntagResource` Required to be able to remove tags from resources.
UpdateServiceLevelObjective	`application-signals:UpdateServiceLevelObjective` Required to update an existing service level objective

CloudWatch Contributor Insights API operations and required permissions for actions

> **Important**
>
> When you grant a user the `cloudwatch:PutInsightRule` permission, by default that user can create a rule that evaluates any log group in CloudWatch Logs. You can add IAM policy conditions that limit these permissions for a user to include and exclude specific log groups. For more information, see Condition keys for Contributor Insights log group access.

CloudWatch Contributor Insights API operations	Required permissions (API actions)
DeleteInsightRules	`cloudwatch:DeleteInsightRules` Required to delete Contributor Insights rules.
DescribeInsightRules	`cloudwatch:DescribeInsightRules` Required to view the Contributor Insights rules in your account.
EnableInsightRules	`cloudwatch:EnableInsightRules` Required to enable Contributor Insights rules.
GetInsightRuleReport	`cloudwatch:GetInsightRuleReport` Required to retrieve time series data and other statistics collectd by Contributor Insights rules.
PutInsightRule	`cloudwatch:PutInsightRule` Required to create Contributor Insights rules. See the **Important** note at the beginning of this table.

CloudWatch Events API operations and required permissions for actions

CloudWatch Events API operations	Required permissions (API actions)
DeleteRule	`events:DeleteRule`

CloudWatch Events API operations	Required permissions (API actions)
	Required to delete a rule.
DescribeRule	`events:DescribeRule` Required to list the details about a rule.
DisableRule	`events:DisableRule` Required to disable a rule.
EnableRule	`events:EnableRule` Required to enable a rule.
ListRuleNamesByTarget	`events:ListRuleNamesByTarget` Required to list rules associated with a target.
ListRules	`events:ListRules` Required to list all rules in your account.
ListTargetsByRule	`events:ListTargetsByRule` Required to list all targets associated with a rule.
PutEvents	`events:PutEvents` Required to add custom events that can be matched to rules.

CloudWatch Events API operations	Required permissions (API actions)
PutRule	`events:PutRule` Required to create or update a rule.
PutTargets	`events:PutTargets` Required to add targets to a rule.
RemoveTargets	`events:RemoveTargets` Required to remove a target from a rule.
TestEventPattern	`events:TestEventPattern` Required to test an event pattern against a given event.

CloudWatch Logs API operations and required permissions for actions

> **Note**
>
> CloudWatch Logs permissions can be found in the CloudWatch Logs user guide.

Amazon EC2 API operations and required permissions for actions

Amazon EC2 API operations	Required permissions (API actions)
DescribeInstanceStatus	`ec2:DescribeInstanceStatus` Required to view EC2 instance status details.
DescribeInstances	`ec2:DescribeInstances`

Amazon EC2 API operations	Required permissions (API actions)
	Required to view EC2 instance details.
RebootInstances	`ec2:RebootInstances` Required to reboot an EC2 instance.
StopInstances	`ec2:StopInstances` Required to stop an EC2 instance.
TerminateInstances	`ec2:TerminateInstances` Required to terminate an EC2 instance.

Amazon EC2 Auto Scaling API operations and required permissions for actions

Amazon EC2 Auto Scaling API operations	Required permissions (API actions)
Scaling	`autoscaling:Scaling` Required to scale an Auto Scaling group.
Trigger	`autoscaling:Trigger` Required to trigger an Auto Scaling action.

Compliance validation for Amazon CloudWatch

Our new AWS sign-up experience is not designed for regulated workloads. If you're using our new AWS sign-up experience, but you want to use AWS for regulated workloads, you can sign up for AWS (advanced) or activate advanced features for your AWS environment.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in Amazon CloudWatch

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Amazon CloudWatch

As a managed service, Amazon CloudWatch is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CloudWatch through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the Amazon Web Services Cloud. A subnet is a range of IP addresses in a VPC. You can deploy a variety of AWS resources in the subnets of your VPCs. For example, you can deploy Amazon EC2 instances, EMR clusters, and DynamoDB tables in subnets. For more information, see the Amazon VPC User Guide.

To enable CloudWatch to communicate with resources in a VPC without going through the public internet, use AWS PrivateLink. For more information, see Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints.

A private subnet is a subnet with no default route to the public internet. Deploying an AWS resource in a private subnet does not prevent Amazon CloudWatch from collecting built-in metrics from the resource.

If you need to publish custom metrics from an AWS resource in a private subnet, you can do so using a proxy server. The proxy server forwards those HTTPS requests to the public API endpoints for CloudWatch.

AWS Security Hub

Monitor your usage of CloudWatch as it relates to security best practices by using AWS Security Hub. Security Hub uses *security controls* to evaluate resource configurations and *security standards* to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate CloudWatch resources, see Amazon CloudWatch controls in the AWS Security Hub User Guide.

Using CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC, CloudWatch, CloudWatch Synthetics, and CloudWatch Network Monitoring features. You can use these connections to enable these services to communicate with resources in your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as the IP address range, subnets, route tables, and network gateways. To connect your VPC to CloudWatch services, you

define an *interface VPC endpoint* for your VPC. The endpoint provides reliable, scalable connectivity to CloudWatch and supported CloudWatch services without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see What is Amazon VPC in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see the following blog post: New – AWS PrivateLink for AWS Services

The following steps are for users of Amazon VPC. For more information, see Getting Started in the *Amazon VPC User Guide*.

CloudWatch VPC endpoints

CloudWatch currently supports VPC endpoints, including IPv6-only and dual stack enabled endpoints, in all AWS Regions, including the AWS GovCloud (US) Regions. For information on endpoints URL, see CloudWatch endpoints and quotas .

Creating a VPC endpoint for CloudWatch

To start using CloudWatch with your VPC, create an interface VPC endpoint for CloudWatch. The service name to choose is `com.amazonaws.`*`region`*`.monitoring`. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch. CloudWatch calls other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch, and you already have metrics flowing to CloudWatch from resources located on your VPC, these metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch. This policy allows users connecting to CloudWatch through the VPC to send metric data to CloudWatch and prevents them from performing other CloudWatch actions.

```
{
  "Statement": [
    {
      "Sid": "PutOnly",
      "Principal": "*",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To edit the VPC endpoint policy for CloudWatch

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch, choose **Create endpoint**. Select **com.amazonaws.*region*.monitoring**, and then choose **Create endpoint**.
4. Select the **com.amazonaws.*region*.monitoring** endpoint, and then choose the **Policy** tab.
5. Choose **Edit policy**, and then make your changes.

CloudWatch Synthetics VPC endpoint

CloudWatch Synthetics currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)

- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)

Creating a VPC endpoint for CloudWatch Synthetics

To start using CloudWatch Synthetics with your VPC, create an interface VPC endpoint for CloudWatch Synthetics. The service name to choose is `com.amazonaws.region.synthetics`. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You do not need to change the settings for CloudWatch Synthetics. CloudWatch Synthetics communicates with other AWS services using either public endpoints or private interface VPC endpoints, whichever are in use. For example, if you create an interface VPC endpoint for CloudWatch Synthetics, and you already have an interface endpoint for Amazon S3, CloudWatch Synthetics begins communicating with Amazon S3 through the interface VPC endpoint by default.

Controlling access to your CloudWatch Synthetics VPC endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, a default policy is attached that allows full access to the service. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies affect canaries that are managed privately by VPC. They are not needed for canaries that run on private subnets.

Endpoint policies must be written in JSON format.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for CloudWatch Synthetics. This policy enables users connecting to CloudWatch Synthetics through the VPC to view information about canaries and their runs, but not to create, modify, or delete canaries.

```
{
    "Statement": [
        {
            "Action": [
                "synthetics:DescribeCanaries",
                "synthetics:GetCanaryRuns"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

To edit the VPC endpoint policy for CloudWatch Synthetics

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for CloudWatch Synthetics, choose **Create endpoint**. Select **com.amazonaws.*region*.synthetics** and then choose **Create endpoint**.
4. Select the **com.amazonaws.*region*.synthetics** endpoint, and then choose the **Policy** tab.
5. Choose **Edit policy**, and then make your changes.

CloudWatch Network Monitoring feature VPC endpoints

CloudWatch Network Monitoring includes the following features: Network Flow Monitor, Internet Monitor, and Network Synthetic Monitor. These features each support VPC endpoints in the AWS Regions where the Network Monitoring feature is supported.

To see a list of supported Regions for each Network Monitoring feature, see the following topics:

- **Network Flow Monitor:** Supported AWS Regions for Network Flow Monitor
- **Internet Monitor:** Supported AWS Regions for Internet Monitor
- **Network Synthetic Monitor:** Supported AWS Regions for Network Synthetic Monitor

Creating a VPC endpoint for a CloudWatch Network Monitoring feature

To start using CloudWatch Network Monitoring features with your VPC, create an interface VPC endpoint for the feature that you want to use. For Network Monitoring, the following service names are available:

- `com.amazonaws.region.networkflowmonitor`
- `com.amazonaws.region.networkflowmonitorreports`
- `com.amazonaws.region.internetmonitor`
- `com.amazonaws.region.internetmonitor-fips`
- `com.amazonaws.region.networkmonitor`

For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

You don't need to change the settings for Network Monitoring services. Network Monitoring services communicate with other AWS services using either public endpoints or private interface VPC endpoints, whichever are being used. For example, if you create an interface VPC endpoint for a Network Monitoring service, and you already have metrics flowing to the service from resources located on your VPC, the metrics begin flowing through the interface VPC endpoint by default.

Controlling access to your CloudWatch Network Monitoring feature VPC endpoints

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. An endpoint policy doesn't override or replace user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

If you don't attach a policy when you create an endpoint, Amazon VPC attaches a default policy for you that allows full access and does not restrict access to a specific service. For additional security, you can attach a policy to the endpoint to specifically limit access to the feature. For example, for Internet Monitor, you could allow full access to just Internet Monitor by attaching the AWS managed policy that enables full access to the feature, CloudWatchInternetMonitorFullAccess. Or, you can further limit permissions to just specific actions for the endpoint.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy that you could create for Network Flow Monitor to limit actions for the endpoint. This policy allows requests to Network Flow Monitor through the VPC to use only the `Publish` action, which enables requests to publish metrics to Network Flow Monitor backend ingestion.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "networkflowmonitor:Publish",
            "Resource": "*"
        }
    ]
}
```

If you want to use a specific VPC endpoint policy with an interface VPC endpoint for a Network Monitoring feature, use steps similar to the following example for adding a policy for Network Flow Monitor.

To edit a VPC endpoint policy for Network Flow Monitor

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for Network Flow Monitor, choose **Create endpoint**.
4. Select **com.amazonaws.*region*.networkflowmonitor**, and then choose **Create endpoint**.
5. Select the **com.amazonaws.*region*.networkflowmonitor** endpoint, and then choose the **Policy** tab.
6. Choose **Edit policy**, and then make your changes.

Security considerations for Synthetics canaries

The following sections explain security issues that you should consider when creating and running canaries in Synthetics.

Use secure connections

Because canary code and the results from canary test runs can contain sensitive information, do not have your canary connect to endpoints over unencrypted connections. Always use encrypted connections, such as those that begin with `https://`.

Canary naming considerations

The Amazon Resource Name (ARN) of a canary is included in the user-agent header as a part of outbound calls made from the Puppeteer-driven Chromium browser that is included as a part of the CloudWatch Synthetics wrapper library. This helps identify CloudWatch Synthetics canary traffic and relate it back to the canaries that are making calls.

The canary ARN includes the canary name. Choose canary names that do not reveal proprietary information.

Additionally, be sure to point your canaries only at websites and endpoints that you control.

Secrets and sensitive information in canary code

If you pass your canary code directly into the canary using a zip file, the contents of the script can be seen in AWS CloudTrail logs.

If you have sensitive information or secrets (such as access keys or database credentials) in a canary script, we strongly recommend that you store the script as a versioned object in Amazon S3 and pass the Amazon S3 location into the canary, instead of passing the canary code by a zip file.

If you do use a zip file to pass the canary script, we strongly recommend that you don't include secrets or sensitive information in your canary source code. For more information about how to use AWS Secrets Manager to help keep your secrets safe, see What is AWS Secrets Manager?.

Permissions considerations

We recommend that you restrict access to resources that are created or used by CloudWatch Synthetics. Use tight permissions on the Amazon S3 buckets where canaries store test run results and other artifacts, such as logs and screenshots.

Similarly, keep tight permissions on the locations where your canary source code is stored, so that no user accidentally or maliciously deletes the Lambda layers or Lambda functions used for the canary.

To help make sure you run the canary code you intend, you can use object versioning on the Amazon S3 bucket where your canary code is stored. Then when you specify this code to run as a canary, you can include the object `versionId` as part of the path, as in the following examples.

```
https://bucket.s3.amazonaws.com/path/object.zip?versionId=version-id
https://s3.amazonaws.com/bucket/path/object.zip?versionId=version-id
https://bucket.s3-region.amazonaws.com/path/object.zip?versionId=version-id
```

Stack traces and exception messages

By default, CloudWatch Synthetics canaries capture any exception thrown by your canary script, no matter whether the script is custom or is from a blueprint. CloudWatch Synthetics logs both the exception message and the stack trace to three locations:

- Back into the CloudWatch Synthetics service to speed up debugging when you describe test runs
- Into CloudWatch Logs according to the configuration that your Lambda functions are created with
- Into the Synthetics log file, which is a plaintext file that is uploaded to the Amazon S3 location specified by the value you set for the `resultsLocation` of the canary

If you want to send and store less information, you can capture exceptions before they return to the CloudWatch Synthetics wrapper library.

You can also have request URLs in your errors. CloudWatch Synthetics scans for any URLs in the error thrown by your script and redacts restricted URL parameters from them based on the **restrictedUrlParameters** configuration. If you are logging error messages in your script, you can use getSanitizedErrorMessage to redact URLs before logging.

Scope your IAM roles narrowly

We recommend that you do not configure your canary to visit potentially malicious URLs or endpoints. Pointing your Canary to untrusted or unknown websites or endpoints could expose your Lambda function code to malicious user's scripts. Assuming a malicious website can break out of Chromium, it could have access to your Lambda code in a similar way to if you connected to it using an internet browser.

Run your Lambda function with an IAM execution role that has scoped-down permissions. This way, if your Lambda function is compromised by a malicious script, it is limited in the actions it can take when running as your canary's AWS account.

When you use the CloudWatch console to create a canary, it is created with a scoped-down IAM execution role.

Sensitive data redaction

CloudWatch Synthetics captures URLs, status code, failure reason (if any), and headers and bodies of requests and responses. This enables a canary user to understand, monitor, and debug canaries.

The configurations described in the following sections can be set at any point in canary execution. You can also choose to apply different configurations to different synthetics steps.

Request URLs

By default, CloudWatch Synthetics logs request URLs, status codes, and the status reason for each URL in canary logs. Request URLs can also appear in canary execution reports, HAR files, and so on. Your request URL might contain sensitive query parameters, such as access tokens or passwords. You can redact sensitive information from being logged by CloudWatch Synthetics.

To redact sensitive information, set the configuration property **restrictedUrlParameters**. For more information, see SyntheticsConfiguration class. This causes CloudWatch Synthetics to redact URL parameters, including path and query parameter values, based on **restrictedUrlParameters** before logging. If you are logging URLs in your script, you can use getSanitizedUrl(url, stepConfig = null) to redact URLs before logging. For more information, see SyntheticsLogHelper class.

Headers

By default, CloudWatch Synthetics doesn't log request/response headers. For UI canaries, this is the default behavior for canaries using runtime version `syn-nodejs-puppeteer-3.2` and later.

If your headers don't contain sensitive information, you can enable headers in HAR file and HTTP reports by setting the **includeRequestHeaders** and **includeResponseHeaders** properties to `true`. You can enable all headers but choose to restrict values of sensitive header keys. For example, you can choose to only redact `Authorization` headers from artifacts produced by canaries.

Request and response body

By default, CloudWatch Synthetics doesn't log the request/response body in canary logs or reports. This information is particularly useful for API canaries. Synthetics captures all HTTP requests and can show headers, request and response bodies. For more information, see executeHttpStep(stepName, requestOptions, [callback], [stepConfig]). You can choose to enable

request/response body by setting the **includeRequestBody** and **includeResponseBody** properties to `true`.

Code examples for CloudWatch using AWS SDKs

The following code examples show how to use CloudWatch with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for CloudWatch using AWS SDKs
  - Hello CloudWatch
  - Learn core operations for CloudWatch using an AWS SDK
  - Actions for CloudWatch using AWS SDKs
    - Use DeleteAlarmMuteRule with an AWS SDK
    - Use DeleteAlarms with an AWS SDK or CLI
    - Use DeleteAnomalyDetector with an AWS SDK or CLI
    - Use DeleteDashboards with an AWS SDK or CLI
    - Use DescribeAlarmContributors with an AWS SDK
    - Use DescribeAlarmHistory with an AWS SDK or CLI
    - Use DescribeAlarms with an AWS SDK or CLI
    - Use DescribeAlarmsForMetric with an AWS SDK or CLI
    - Use DescribeAnomalyDetectors with an AWS SDK or CLI
    - Use DisableAlarmActions with an AWS SDK or CLI
    - Use EnableAlarmActions with an AWS SDK or CLI
    - Use GetAlarmMuteRule with an AWS SDK
    - Use GetDashboard with an AWS SDK or CLI

- Use GetMetricData with an AWS SDK or CLI
- Use GetMetricStatistics with an AWS SDK or CLI
- Use GetMetricWidgetImage with an AWS SDK or CLI
- Use GetOTelEnrichment with an AWS SDK
- Use ListAlarmMuteRules with an AWS SDK
- Use ListDashboards with an AWS SDK or CLI
- Use ListMetrics with an AWS SDK or CLI
- Use PutAlarmMuteRule with an AWS SDK
- Use PutAnomalyDetector with an AWS SDK or CLI
- Use PutDashboard with an AWS SDK or CLI
- Use PutMetricAlarm with an AWS SDK or CLI
- Use PutMetricData with an AWS SDK or CLI
- Use StartOTelEnrichment with an AWS SDK
- Use StopOTelEnrichment with an AWS SDK

- Scenarios for CloudWatch using AWS SDKs
  - Creating a monitoring dashboard with function name as a variable
  - Get started with CloudWatch alarms using an AWS SDK
  - Manage CloudWatch custom metrics and alarms using an AWS SDK
  - Monitor performance of Amazon DynamoDB using an AWS SDK
  - Run CPU stress tests on virtual machine instances using fault injection
  - Send OpenTelemetry metrics to CloudWatch and alarm on them with PromQL using an AWS SDK
  - Using property variables in monitoring dashboards to monitor multiple serverless functions

Basic examples for CloudWatch using AWS SDKs

The following code examples show how to use the basics of Amazon CloudWatch with AWS SDKs.

Examples

- Hello CloudWatch
- Learn core operations for CloudWatch using an AWS SDK

- Actions for CloudWatch using AWS SDKs
  - Use DeleteAlarmMuteRule with an AWS SDK
  - Use DeleteAlarms with an AWS SDK or CLI
  - Use DeleteAnomalyDetector with an AWS SDK or CLI
  - Use DeleteDashboards with an AWS SDK or CLI
  - Use DescribeAlarmContributors with an AWS SDK
  - Use DescribeAlarmHistory with an AWS SDK or CLI
  - Use DescribeAlarms with an AWS SDK or CLI
  - Use DescribeAlarmsForMetric with an AWS SDK or CLI
  - Use DescribeAnomalyDetectors with an AWS SDK or CLI
  - Use DisableAlarmActions with an AWS SDK or CLI
  - Use EnableAlarmActions with an AWS SDK or CLI
  - Use GetAlarmMuteRule with an AWS SDK
  - Use GetDashboard with an AWS SDK or CLI
  - Use GetMetricData with an AWS SDK or CLI
  - Use GetMetricStatistics with an AWS SDK or CLI
  - Use GetMetricWidgetImage with an AWS SDK or CLI
  - Use GetOTelEnrichment with an AWS SDK
  - Use ListAlarmMuteRules with an AWS SDK
  - Use ListDashboards with an AWS SDK or CLI
  - Use ListMetrics with an AWS SDK or CLI
  - Use PutAlarmMuteRule with an AWS SDK
  - Use PutAnomalyDetector with an AWS SDK or CLI
  - Use PutDashboard with an AWS SDK or CLI
  - Use PutMetricAlarm with an AWS SDK or CLI
  - Use PutMetricData with an AWS SDK or CLI
  - Use StartOTelEnrichment with an AWS SDK
  - Use StopOTelEnrichment with an AWS SDK

Hello CloudWatch

The following code examples show how to get started using CloudWatch.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
```

```
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m
 => $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}
```

- For API details, see ListMetrics in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import software.amazon.awssdk.services.cloudwatch.model.ListMetricsRequest;
import software.amazon.awssdk.services.cloudwatch.paginators.ListMetricsIterable;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloService {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <namespace>\s

                Where:
                  namespace - The namespace to filter against (for example, AWS/
EC2).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String namespace = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        listMets(cw, namespace);
        cw.close();
    }

    public static void listMets(CloudWatchClient cw, String namespace) {
        try {
            ListMetricsRequest request = ListMetricsRequest.builder()
                    .namespace(namespace)
                    .build();

            ListMetricsIterable listRes = cw.listMetricsPaginator(request);
            listRes.stream()
                    .flatMap(r -> r.metrics().stream())
```

```
                    .forEach(metrics -> System.out.println(" Retrieved metric is:
 " + metrics.metricName()));

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListMetrics in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
Before running this Kotlin code example, set up your development environment,
including your credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
 */
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
           <namespace>
        Where:
           namespace - The namespace to filter against (for example, AWS/EC2).
    """

    if (args.size != 1) {
        println(usage)
        exitProcess(0)
```

```
    }

    val namespace = args[0]
    listAllMets(namespace)
}

suspend fun listAllMets(namespaceVal: String?) {
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listMetricsPaginated(request)
            .transform { it.metrics?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.metricName}")
                println("Namespace is ${obj.namespace}")
            }
    }
}
```

- For API details, see ListMetrics in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn core operations for CloudWatch using an AWS SDK

The following code examples show how to:

- List CloudWatch namespaces and metrics.
- Start OpenTelemetry enrichment so CloudWatch correlates incoming OTLP metrics with the resources that produced them.
- See how OTLP metrics reach the CloudWatch metrics endpoint. Metric ingestion over OTLP is not an AWS SDK operation.
- Create an alarm that evaluates a PromQL query.

- Inspect the alarm's contributors, the individual series that the query matched.
- Get statistics for a metric and chart it on a dashboard.
- Mute the alarm for a maintenance window, then clean up.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This scenario demonstrates the Amazon CloudWatch OpenTelemetry (OTel)
 experience.
    CloudWatch ingests OpenTelemetry metrics natively, and this example walks
 through what
    you do with them: turning on enrichment so CloudWatch can correlate incoming
 OTLP
    metrics with the resources that produced them, alarming on those metrics with
 a PromQL
    query, and finding out which individual series drove the alarm.

    A PromQL alarm works differently from a classic metric alarm. Rather than
 watching one
    metric and counting breaching periods, it evaluates a query that can match
 many series
    at once, and tracks each matching series separately as a contributor.

    Note that sending OTLP metrics to CloudWatch is not an AWS SDK operation.
 Metrics
    arrive over the OTLP protocol through the CloudWatch agent, an OpenTelemetry
```

```
   Collector, or an ADOT SDK. Everything this scenario does is configuration and
querying
   around that ingestion path.

   This .NET example performs the following tasks:
       1. List metrics and namespaces from CloudWatch.
       2. Start OpenTelemetry enrichment for the account.
       3. Explain how OTLP metrics reach CloudWatch.
       4. Create an alarm that evaluates a PromQL query.
       5. Inspect the contributors to the PromQL alarm.
       6. Get metric statistics and chart the metric on a dashboard.
       7. Mute the alarm for a maintenance window.
       8. Clean up resources.
   */

   private static ILogger logger = null!;
   private static CloudWatchWrapper _cloudWatchWrapper = null!;
   private static CloudWatchOTelWrapper _otelWrapper = null!;
   private static IConfiguration _configuration = null!;

   private const string DefaultQuery = "avg by (host) (system_cpu_utilization) >
80";

   // Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to
3600 seconds.
   private const int EvaluationInterval = 60;
   private const int PendingPeriod = 300;
   private const int RecoveryPeriod = 120;

   private static string _alarmName = null!;
   private static string _dashboardName = null!;
   private static string _muteRuleName = null!;

   // Tracks whether this run turned enrichment on, so that cleanup only turns
off
   // enrichment that this run started.
   private static bool _startedEnrichment;
   private static bool _dashboardCreated;
   private static string _region = null!;

   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon service.
       using var host = Host.CreateDefaultBuilder(args)
```

```
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
            .AddTransient<CloudWatchWrapper>()
            .AddTransient<CloudWatchOTelWrapper>()
        )
        .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();
        _otelWrapper = host.Services.GetRequiredService<CloudWatchOTelWrapper>();

        // A metric widget must name its region, because a dashboard can chart
        // metrics from several.
        _region = host.Services.GetRequiredService<IAmazonCloudWatch>()
            .Config.RegionEndpoint.SystemName;

        // Suffix the resource names so repeated runs do not collide.
        var suffix = Random.Shared.Next(1000, 9999).ToString();
        _alarmName = $"doc-example-promql-alarm-{suffix}";
        _dashboardName = $"doc-example-dashboard-{suffix}";
        _muteRuleName = $"doc-example-mute-rule-{suffix}";

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch Basics scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine(
            "\nCloudWatch now ingests OpenTelemetry metrics natively. This
scenario walks through" +
```

```
            "\nthat experience: it turns on OTel enrichment so CloudWatch can
 correlate incoming" +
            "\nOTLP metrics with the resources that produced them, alarms on
 those metrics with a" +
            "\nPromQL query, and shows you which individual series drove the
 alarm." +
            "\n" +
            "\nA PromQL alarm works differently from a classic metric alarm.
 Rather than watching" +
            "\none metric and counting breaching periods, it evaluates a query
 that can match many" +
            "\nseries at once, and tracks each one separately as a contributor.
\n");

        try
        {
            var namespaces = await ListMetricsAndNamespaces();
            await StartOTelEnrichment();
            ExplainOtlpIngestion();
            await CreatePromQlAlarm();
            await InspectAlarmContributors();
            await GetStatisticsAndChartMetric(namespaces);
            await MuteAlarmForMaintenance();
            await CleanUp();

            Console.WriteLine(new string('-', 80));
            Console.WriteLine("CloudWatch Basics scenario is complete.");
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            Console.WriteLine(new string('-', 80));
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanUp();
            Console.WriteLine(new string('-', 80));
        }
    }

    /// <summary>
    /// List the metrics and namespaces already present in the account, to orient
 the
    /// reader before any configuration happens.
    /// </summary>
    /// <returns>The distinct namespaces found.</returns>
```

```
    private static async Task<List<string>> ListMetricsAndNamespaces()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("1. List metrics and namespaces");
        Console.WriteLine(
            "\nBefore configuring anything, let's see what CloudWatch is already
collecting in" +
            "\nthis account by calling ListMetrics.\n");

        var metrics = await _cloudWatchWrapper.ListMetrics();

        // Order by metric count descending, so the most-populated namespace
comes first.
        // Step 6 charts a metric from that namespace, and a busy namespace is
the one most
        // likely to have datapoints worth looking at.
        var namespaceCounts = metrics
            .GroupBy(m => m.Namespace)
            .Select(g => new { Namespace = g.Key, Count = g.Count() })
            .OrderByDescending(n => n.Count)
            .ToList();
        var namespaces = namespaceCounts.Select(n => n.Namespace).ToList();

        Console.WriteLine($"\tFound {metrics.Count} metrics across
{namespaces.Count} namespaces:");
        foreach (var entry in namespaceCounts.Take(10))
        {
            Console.WriteLine($"\t  {entry.Namespace} ({entry.Count} metrics)");
        }

        if (!namespaces.Any())
        {
            Console.WriteLine(
                "\tNo metrics found in this account. The statistics and dashboard
steps later on" +
                "\n\tneed an existing metric, so they will be skipped.");
        }

        Console.WriteLine(new string('-', 80));
        return namespaces;
    }

    /// <summary>
```

```
   /// Start OTel enrichment, but only if it is not already running. Enrichment
is what
   /// makes CloudWatch attach AWS resource context to incoming OTLP metrics.
   /// </summary>
   private static async Task StartOTelEnrichment()
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine("2. Start OpenTelemetry enrichment");
       Console.WriteLine(
           "\nEnrichment is what lets CloudWatch attach AWS resource context to
the OTLP metrics" +
           "\nyou send it. Without it, your metrics arrive as opaque series with
no connection to" +
           "\nthe resources that emitted them." +
           "\n" +
           "\nWe check the current state first, and only start enrichment if it
isn't already on.\n");

       var status = await _otelWrapper.GetOTelEnrichmentStatus();
       Console.WriteLine($"\tEnrichment status: {status}");

       if (status != OTelEnrichmentStatus.Running)
       {
           // Record the attempt before making it. We already know enrichment
was not running,
           // so stopping it during cleanup is always safe, and a call that
starts enrichment
           // but then fails to report back (a timeout, say) would otherwise
leave it running.
           _startedEnrichment = true;
           await _otelWrapper.StartOTelEnrichment();

           status = await _otelWrapper.GetOTelEnrichmentStatus();
           Console.WriteLine($"\tEnrichment status: {status}");
           Console.WriteLine(
               "\n\tNote: this run started enrichment, so the cleanup step will
stop it again.");
       }
       else
       {
           Console.WriteLine(
               "\n\tEnrichment was already running, so we will leave it alone.
The cleanup step" +
```

```
                "\n\twill not stop it, because other workloads in this account
may depend on it.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Explain that OTLP metric ingestion is not an AWS SDK operation. This step
makes no
    /// service call; naming the gap explicitly is the point.
    /// </summary>
    private static void ExplainOtlpIngestion()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("3. Send OTLP metrics to CloudWatch");
        Console.WriteLine(
            "\nThis step is not an AWS SDK operation, and that's worth being
explicit about." +
            "\nMetrics reach CloudWatch over the OTLP protocol, through the
CloudWatch agent, an" +
            "\nOpenTelemetry Collector, or an ADOT SDK. There is no
PutOTelMetrics API to call." +
            "\n" +
            "\nPoint your collector at the CloudWatch metrics endpoint, which
follows the pattern" +
            "\n\thttps://monitoring.<region>.amazonaws.com/v1/metrics" +
            "\n" +
            "\nThe endpoint is HTTP/1.1 only and does not support gRPC, so use an
otlphttp" +
            "\nexporter rather than otlp. The metrics endpoint signs as
\"monitoring\".\n");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create an alarm whose evaluation is a PromQL query.
    /// </summary>
    private static async Task CreatePromQlAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Create a PromQL alarm");
        Console.WriteLine(
```

```
            "\nNow we alarm on those metrics. The comparison goes inside the
query itself: a" +
            "\nPromQL alarm has no separate threshold, comparison operator,
statistic, or period.\n");

        Console.WriteLine($"Enter a PromQL query, or press <ENTER> for the
default\n[{DefaultQuery}]:");
        var input = Console.ReadLine();
        var query = string.IsNullOrWhiteSpace(input) ? DefaultQuery :
input.Trim();

        await _otelWrapper.PutPromQLMetricAlarm(_alarmName, query,
            EvaluationInterval, PendingPeriod, RecoveryPeriod);

        Console.WriteLine($"\tCreated alarm {_alarmName}:");
        Console.WriteLine($"\t  query:              {query}");
        Console.WriteLine($"\t  evaluationInterval: {EvaluationInterval}
seconds");
        Console.WriteLine($"\t  pendingPeriod:      {PendingPeriod} seconds");
        Console.WriteLine($"\t  recoveryPeriod:     {RecoveryPeriod} seconds");
        Console.WriteLine(
            "\n\tA PromQL alarm starts in the OK state rather than
INSUFFICIENT_DATA, which is" +
            "\n\tanother way it differs from a classic alarm.");

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Show which individual series the alarm's query matched. This is the step
with no
    /// classic-alarm equivalent.
    /// </summary>
    private static async Task InspectAlarmContributors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Inspect the alarm's contributors");
        Console.WriteLine(
            "\nEach contributor is one series the query matched, identified by
its label set." +
            "\nThis is how you find out which host is unhealthy rather than only
that something" +
            "\nis. Classic alarms have no equivalent.\n");
```

```
        var contributors = await
_otelWrapper.DescribeAlarmContributors(_alarmName);

        if (!contributors.Any())
        {
            Console.WriteLine(
                "\tNo contributors yet. The query matched no series, which
usually means no OTel" +
                "\n\tmetrics with these labels have arrived. Once your collector
is sending data," +
                "\n\teach matching series appears here with its labels and why it
breached.");
        }
        else
        {
            Console.WriteLine($"\tFound {contributors.Count} contributors:");
            foreach (var contributor in contributors)
            {
                var labels = string.Join(", ",
                    contributor.ContributorAttributes.Select(a =>
$"{a.Key}={a.Value}"));
                Console.WriteLine($"\t  {contributor.ContributorId}: {labels}");
                Console.WriteLine($"\t    reason: {contributor.StateReason}");
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get statistics for an existing metric and chart it on a dashboard, so the
reader can
    /// see what the alarm is evaluating.
    /// </summary>
    /// <param name="namespaces">The namespaces discovered in step 1.</param>
    private static async Task GetStatisticsAndChartMetric(List<string>
namespaces)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("6. Get statistics and chart the metric on a
dashboard");
        Console.WriteLine("\nStatistics and dashboards are how you see what the
alarm is evaluating.\n");
```

```
        if (!namespaces.Any())
        {
            Console.WriteLine("\tSkipping statistics and dashboard because no
metrics exist yet.");
            Console.WriteLine(new string('-', 80));
            return;
        }

        var metricNamespace = namespaces.First();
        var metrics = await _cloudWatchWrapper.ListMetrics(metricNamespace);
        var metric = metrics.FirstOrDefault();

        if (metric != null)
        {
            var datapoints = await _cloudWatchWrapper.GetMetricStatistics(
                metricNamespace, metric.MetricName, new List<string> { "Average",
"Maximum" },
                metric.Dimensions, 1, 3600);

            Console.WriteLine(
                $"\tStatistics for {metricNamespace} {metric.MetricName} over the
last day:");
            Console.WriteLine($"\t  Datapoints: {datapoints.Count}");
            foreach (var datapoint in datapoints.Take(3))
            {
                Console.WriteLine(
                    $"\t  {datapoint.Timestamp:u} average {datapoint.Average},
maximum {datapoint.Maximum}");
            }

            var dashboardBody = BuildDashboardBody(metricNamespace, metric,
_region);
            var validationMessages = await
_cloudWatchWrapper.PutDashboard(_dashboardName, dashboardBody);
            _dashboardCreated = true;

            if (validationMessages.Any())
            {
                foreach (var message in validationMessages)
                {
                    Console.WriteLine($"\tDashboard validation message:
{message.Message}");
                }
            }
```

```
            Console.WriteLine($"\tCreated dashboard {_dashboardName}.");

            var dashboard = await
 _cloudWatchWrapper.GetDashboard(_dashboardName);
            Console.WriteLine($"\tRead the dashboard back, {dashboard.Length}
 characters of widget JSON.");
        }
        else
        {
            Console.WriteLine($"\tNo metrics found in namespace
 {metricNamespace}, skipping.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Build a single-widget dashboard body that charts the given metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric to chart.</
param>
    /// <param name="metric">The metric to chart.</param>
    /// <param name="region">The region the metric is in. A metric widget must
 name its
    /// region, because a dashboard can chart metrics from several.</param>
    internal static string BuildDashboardBody(string metricNamespace, Metric
 metric, string region)
    {
        var dimensionParts = string.Concat(
            metric.Dimensions.Select(d => $", \"{d.Name}\", \"{d.Value}\""));

        return $@"{{
    ""widgets"": [
        {{
            ""type"": ""text"",
            ""x"": 0, ""y"": 0, ""width"": 24, ""height"": 2,
            ""properties"": {{
                ""markdown"": ""This dashboard was created programmatically by an
 AWS SDK code example.""
            }}
        }},
        {{
            ""type"": ""metric"",
```

```
            ""x"": 0, ""y"": 2, ""width"": 12, ""height"": 6,
            ""properties"": {{
                ""metrics"": [[ ""{metricNamespace}"",
 ""{metric.MetricName}""{dimensionParts} ]],
                ""view"": ""timeSeries"",
                ""stat"": ""Average"",
                ""period"": 300,
                ""region"": ""{region}"",
                ""title"": ""{metric.MetricName}""
            }}
        }}
    ]
}}";
    }

    /// <summary>
    /// Create a mute rule so the alarm's actions are suppressed during a
 maintenance
    /// window, then read it back and find it in the account's rules.
    /// </summary>
    private static async Task MuteAlarmForMaintenance()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("7. Mute the alarm for a maintenance window");
        Console.WriteLine(
            "\nWhile a mute rule is active the targeted alarms keep evaluating
 and keep changing" +
            "\nstate, but their actions do not fire. This is the supported way to
 suppress" +
            "\nnotifications during planned maintenance, instead of disabling
 alarm actions and" +
            "\nhoping someone remembers to turn them back on.\n");

        // The expression is a five-field cron expression,
        // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
 five fields,
        // not the six that Amazon EventBridge uses. For a one-time window, use
        // at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
 duration from
        // PT1M to P15D, so PT2H rather than 2h.
        const string expression = "cron(0 2 * * SUN)";
        const string duration = "PT2H";
        const string timezone = "America/Los_Angeles";
```

```
        await _otelWrapper.PutAlarmMuteRule(_muteRuleName, expression, duration,
timezone,
            new List<string> { _alarmName });

        Console.WriteLine($"\tCreated mute rule {_muteRuleName}:");
        Console.WriteLine($"\t  schedule: {expression} for {duration}");
        Console.WriteLine($"\t  timezone: {timezone}");
        Console.WriteLine($"\t  targets:  {_alarmName}");
        Console.WriteLine(
            "\n\tNote the two formats here. The expression is a five-field cron
expression, five" +
            "\n\trather than the six Amazon EventBridge uses. The duration is an
ISO 8601" +
            "\n\tduration, so 'PT2H' and not '2h'." +
            "\n" +
            "\n\tAlso note that MuteTargets is set explicitly. If you leave it
out, the rule" +
            "\n\tapplies to every alarm in the account.");

        var muteRule = await _otelWrapper.GetAlarmMuteRule(_muteRuleName);
        Console.WriteLine(
            $"\tRead the rule back: status {muteRule.Status}, mute type
{muteRule.MuteType}.");

        var summaries = await _otelWrapper.ListAlarmMuteRules(_alarmName);
        Console.WriteLine($"\tFound {summaries.Count} mute rules targeting this
alarm.");

        // Mute rule summaries carry no name field, only an ARN, so match on the
ARN suffix.
        var match = summaries.FirstOrDefault(s =>
            s.AlarmMuteRuleArn.EndsWith($"/{_muteRuleName}") ||
            s.AlarmMuteRuleArn.EndsWith($":{_muteRuleName}"));

        if (match != null)
        {
            Console.WriteLine($"\t  matched by ARN: {match.AlarmMuteRuleArn}
({match.Status})");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
```

```
   /// Delete the resources the scenario created. Each deletion is attempted
independently
   /// so that one failure does not leave the remaining resources behind.
   /// </summary>
   private static async Task CleanUp()
   {
       Console.WriteLine(new string('-', 80));
       Console.WriteLine("8. Clean up");
       Console.WriteLine("\nDelete the resources this scenario created? (y/n)");

       var response = Console.ReadLine();
       if (!string.Equals(response?.Trim(), "y",
StringComparison.OrdinalIgnoreCase))
       {
           Console.WriteLine(
               "\tSkipping cleanup. Note that the alarm, dashboard, and mute
rule are still in" +
               "\n\tyour account, and enrichment may still be running.");
           Console.WriteLine(new string('-', 80));
           return;
       }

       try
       {
           await _otelWrapper.DeleteAlarmMuteRule(_muteRuleName);
           Console.WriteLine($"\tDeleted mute rule {_muteRuleName}.");
       }
       catch (Exception ex)
       {
           Console.WriteLine($"\tCould not delete the mute rule: {ex.Message}");
       }

       try
       {
           await _cloudWatchWrapper.DeleteAlarms(new List<string>
{ _alarmName });
           Console.WriteLine($"\tDeleted alarm {_alarmName}.");
       }
       catch (Exception ex)
       {
           Console.WriteLine($"\tCould not delete the alarm: {ex.Message}");
       }

       if (_dashboardCreated)
```

```
        {
            try
            {
                await _cloudWatchWrapper.DeleteDashboards(new List<string>
 { _dashboardName });
                Console.WriteLine($"\tDeleted dashboard {_dashboardName}.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"\tCould not delete the dashboard:
 {ex.Message}");
            }
        }

        if (_startedEnrichment)
        {
            try
            {
                await _otelWrapper.StopOTelEnrichment();
                Console.WriteLine("\tStopped OTel enrichment, because this run
 started it.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"\tCould not stop OTel enrichment:
 {ex.Message}");
            }
        }
        else
        {
            Console.WriteLine(
                "\tLeft OTel enrichment running, because it was already on before
 this run.");
        }

        Console.WriteLine(new string('-', 80));
    }
}
```

Wrapper methods used by the scenario for the CloudWatch OpenTelemetry actions.

```
/// <summary>
```

```
/// Wrapper class for the OpenTelemetry features of Amazon CloudWatch: turning on
 OTel
/// enrichment so that CloudWatch vended metrics are queryable with PromQL,
 alarming on a
/// PromQL query, inspecting the individual series (contributors) that put a
 PromQL alarm
/// into ALARM, and muting alarm actions on a schedule.
///
/// Note that OTLP metric ingestion is not an AWS SDK operation. To send
 OpenTelemetry
/// metrics to CloudWatch, point an OpenTelemetry collector or the AWS Distro for
/// OpenTelemetry (ADOT) SDK at the CloudWatch OTLP metrics endpoint,
/// https://monitoring.{region}.amazonaws.com/v1/metrics. The operations here
 cover
/// everything you do after those metrics land in CloudWatch.
/// </summary>
public class CloudWatchOTelWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchOTelWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch OpenTelemetry wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchOTelWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchOTelWrapper> logger)
    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }
```

Wrapper methods used by the scenario for the metric, statistic, and dashboard actions.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
```

```
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

```
    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
 metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
 int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTime = DateTime.UtcNow.AddDays(-days),
                EndTime = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
```

```
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }
```

```
        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };
```

```
        var metricImageWidgetString =
 JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
```

```
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTime = startTimeUtc,
                EndTime = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
```

```
   /// <param name="threshold">The threshold value for the alarm.</param>
   /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
       string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
   {
       try
       {
           var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
               new PutMetricAlarmRequest()
               {
                   AlarmActions = alarmActions,
                   AlarmDescription = alarmDescription,
                   AlarmName = alarmName,
                   ComparisonOperator = comparison,
                   Threshold = threshold,
                   Namespace = metricNamespace,
                   MetricName = metricName,
                   EvaluationPeriods = 1,
                   Period = 10,
                   Statistic = new Statistic("Maximum"),
                   DatapointsToAlarm = 1,
                   TreatMissingData = "ignore"
               });
           return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (LimitExceededException lex)
       {
           _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
       }

       return false;
   }

   /// <summary>
   /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
   /// </summary>
   /// <param name="accountId">The AccountId for the alarm.</param>
```

```
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
 topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
 append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
 null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
 _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
```

```
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDate = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDate = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
```

```
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });
```

```
        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }
```

```
        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
 _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Example settings.json values for the scenario.

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
 example code.\n"
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
            "left": {
```

```
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
        "type": "metric",
        "properties": {
          "metrics": [
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
"Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
```

```
        }
      ]
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetDashboard
  - GetMetricStatistics
  - GetOTelEnrichment
  - ListAlarmMuteRules
  - ListDashboards
  - ListMetrics
  - PutAlarmMuteRule
  - PutDashboard
  - PutMetricAlarm
  - StartOTelEnrichment
  - StopOTelEnrichment

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/AlarmPromQLCriteria.h>
#include <aws/monitoring/model/DeleteAlarmMuteRuleRequest.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <aws/monitoring/model/DeleteDashboardsRequest.h>
#include <aws/monitoring/model/DescribeAlarmContributorsRequest.h>
#include <aws/monitoring/model/EvaluationCriteria.h>
#include <aws/monitoring/model/GetAlarmMuteRuleRequest.h>
#include <aws/monitoring/model/GetDashboardRequest.h>
#include <aws/monitoring/model/GetMetricStatisticsRequest.h>
#include <aws/monitoring/model/GetOTelEnrichmentRequest.h>
#include <aws/monitoring/model/ListAlarmMuteRulesRequest.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/MuteTargets.h>
#include <aws/monitoring/model/PutAlarmMuteRuleRequest.h>
#include <aws/monitoring/model/PutDashboardRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <aws/monitoring/model/Rule.h>
#include <aws/monitoring/model/Schedule.h>
#include <aws/monitoring/model/StartOTelEnrichmentRequest.h>
#include <aws/monitoring/model/StopOTelEnrichmentRequest.h>
#include <chrono>
#include <iostream>
#include <map>
#include <random>

namespace {
    const Aws::String DASHES(80, '-');
    const char DEFAULT_QUERY[] = "avg by (host) (system_cpu_utilization) > 80";

    // Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to
 3600
    // seconds.
    const int EVALUATION_INTERVAL = 60;
    const int PENDING_PERIOD = 300;
    const int RECOVERY_PERIOD = 120;

    //! Wait for the reader before moving to the next step.
    void pressEnter() {
        std::cout << "Press Enter to continue..." << std::endl;
```

```
        std::cin.get();
    }

    //! List the account's metrics and report how they are spread across
namespaces.
    /*!
      \param client: A CloudWatch client.
      \param metric: Receives the first metric found, for later steps to chart.
      \return bool: Function succeeded.
     */
    bool listMetricsAndNamespaces(const Aws::CloudWatch::CloudWatchClient
&client,
                                  Aws::CloudWatch::Model::Metric &metric) {
        std::cout << "1. List metrics and namespaces" << std::endl << std::endl;
        std::cout << "Before configuring anything, let's see what CloudWatch is
already"
                  << std::endl
                  << "collecting in this account." << std::endl << std::endl;

        std::map<Aws::String, int> counts;
        int metricCount = 0;
        bool haveMetric = false;

        Aws::CloudWatch::Model::ListMetricsRequest request;
        bool done = false;
        while (!done) {
            auto outcome = client.ListMetrics(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to list metrics: " <<
outcome.GetError().GetMessage()
                          << std::endl;
                return false;
            }

            for (const auto &found : outcome.GetResult().GetMetrics()) {
                ++counts[found.GetNamespace()];
                ++metricCount;
                if (!haveMetric) {
                    metric = found;
                    haveMetric = true;
                }
            }

            const auto &nextToken = outcome.GetResult().GetNextToken();
```

```
            request.SetNextToken(nextToken);
            // This account may have a very large number of metrics, so stop once
there
            // are enough to give the reader a sense of what is there.
            done = nextToken.empty() || metricCount >= 500;
        }

        std::cout << "Found " << metricCount << " metrics across " <<
counts.size()
                  << " namespaces:" << std::endl;
        for (const auto &entry : counts) {
            std::cout << "  " << entry.first << " (" << entry.second << "
metrics)"
                      << std::endl;
        }
        if (!haveMetric) {
            std::cout << "No metrics found in this account. The statistics and
dashboard"
                      << std::endl
                      << "steps later on need an existing metric, so they will be
"
                         "skipped."
                      << std::endl;
        }

        return true;
    }

    //! Start OTel enrichment, but only if it is not already running.
    /*!
      \param client: A CloudWatch client.
      \param startedEnrichment: Set to true if this run started enrichment, so
that
             cleanup only stops what this run turned on.
      \return bool: Function succeeded.
     */
    bool startOTelEnrichment(const Aws::CloudWatch::CloudWatchClient &client,
                             bool &startedEnrichment) {
        std::cout << "2. Start OpenTelemetry enrichment" << std::endl <<
std::endl;
        std::cout << "Enrichment is what lets CloudWatch attach AWS resource
context to"
                  << std::endl
```

```
                  << "the OTLP metrics you send it. Without it, your metrics
arrive as"
                  << std::endl
                  << "opaque series with no connection to the resources that
emitted "
                     "them."
                  << std::endl << std::endl;

       Aws::CloudWatch::Model::GetOTelEnrichmentRequest getRequest;
       auto getOutcome = client.GetOTelEnrichment(getRequest);
       if (!getOutcome.IsSuccess()) {
           std::cerr << "Failed to get OTel enrichment status: "
                     << getOutcome.GetError().GetMessage() << std::endl;
           return false;
       }

       auto status = getOutcome.GetResult().GetStatus();
       std::cout << "Enrichment status: "
                 << Aws::CloudWatch::Model::OTelEnrichmentStatusMapper::
                        GetNameForOTelEnrichmentStatus(status)
                 << std::endl;

       if (status == Aws::CloudWatch::Model::OTelEnrichmentStatus::Running) {
           std::cout << std::endl
                     << "Enrichment was already running, so it will be left
alone. The"
                     << std::endl
                     << "cleanup step will not stop it, because other workloads
in this"
                     << std::endl
                     << "account may depend on it." << std::endl;
           return true;
       }

       Aws::CloudWatch::Model::StartOTelEnrichmentRequest startRequest;
       auto startOutcome = client.StartOTelEnrichment(startRequest);
       if (!startOutcome.IsSuccess()) {
           std::cerr << "Failed to start OTel enrichment: "
                     << startOutcome.GetError().GetMessage() << std::endl;
           return false;
       }
       startedEnrichment = true;

       auto afterOutcome = client.GetOTelEnrichment(getRequest);
```

```
        if (afterOutcome.IsSuccess()) {
            std::cout << "Enrichment status: "
                      << Aws::CloudWatch::Model::OTelEnrichmentStatusMapper::
                             GetNameForOTelEnrichmentStatus(
                                 afterOutcome.GetResult().GetStatus())
                      << std::endl;
        }
        std::cout << std::endl
                  << "Note: this run started enrichment, so the cleanup step will
stop it"
                  << std::endl
                  << "again." << std::endl;

        return true;
    }

    //! Explain how OTLP metrics reach CloudWatch. There is no SDK call for this
step.
    void explainOtlpIngestion() {
        std::cout << "3. Send OTLP metrics to CloudWatch" << std::endl <<
std::endl;
        std::cout << "This step is not an AWS SDK operation, and that is worth
being"
                  << std::endl
                  << "explicit about. Metrics reach CloudWatch over the OTLP
protocol,"
                  << std::endl
                  << "through the CloudWatch agent, an OpenTelemetry Collector,
or an "
                     "ADOT"
                  << std::endl
                  << "SDK. There is no PutOTelMetrics API to call." << std::endl
                  << std::endl;
        std::cout << "Point your collector at the CloudWatch metrics endpoint,
which"
                  << std::endl
                  << "follows the pattern" << std::endl
                  << "  https://monitoring.<region>.amazonaws.com/v1/metrics" <<
std::endl
                  << std::endl;
        std::cout << "The endpoint is HTTP/1.1 only and does not support gRPC, so
use an"
                  << std::endl
```

```
                  << "otlphttp exporter rather than otlp. The metrics endpoint
signs as"
                  << std::endl
                  << "\"monitoring\"." << std::endl;
   }

   //! Create an alarm that evaluates a PromQL query.
   /*!
     \param client: A CloudWatch client.
     \param alarmName: The name of the alarm to create.
     \param query: The PromQL query to evaluate.
     \return bool: Function succeeded.
    */
   bool createPromQLAlarm(const Aws::CloudWatch::CloudWatchClient &client,
                          const Aws::String &alarmName, const Aws::String
&query) {
       std::cout << "4. Create a PromQL alarm" << std::endl << std::endl;
       std::cout << "The comparison goes inside the query itself: a PromQL alarm
has no"
                 << std::endl
                 << "separate threshold, comparison operator, statistic, or
period."
                 << std::endl << std::endl;

       Aws::CloudWatch::Model::AlarmPromQLCriteria promQLCriteria;
       promQLCriteria.SetQuery(query);
       promQLCriteria.SetPendingPeriod(PENDING_PERIOD);
       promQLCriteria.SetRecoveryPeriod(RECOVERY_PERIOD);

       Aws::CloudWatch::Model::EvaluationCriteria evaluationCriteria;
       evaluationCriteria.SetPromQLCriteria(promQLCriteria);

       // EvaluationCriteria is mutually exclusive with the classic MetricName
and
       // Metrics fields. When you use it you must also set EvaluationInterval,
and you
       // must not set Period, Statistic, Threshold, ComparisonOperator,
       // EvaluationPeriods, DatapointsToAlarm, or TreatMissingData.
       Aws::CloudWatch::Model::PutMetricAlarmRequest request;
       request.SetAlarmName(alarmName);
       request.SetAlarmDescription(
           "A PromQL alarm created by the AWS SDK for C++ Basics scenario.");
       request.SetEvaluationCriteria(evaluationCriteria);
       request.SetEvaluationInterval(EVALUATION_INTERVAL);
```

```
        request.SetActionsEnabled(false);

        auto outcome = client.PutMetricAlarm(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to create PromQL alarm: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }

        std::cout << "Created alarm " << alarmName << ":" << std::endl;
        std::cout << "  query:              " << query << std::endl;
        std::cout << "  evaluationInterval: " << EVALUATION_INTERVAL << "
seconds"
                  << std::endl;
        std::cout << "  pendingPeriod:      " << PENDING_PERIOD << " seconds" <<
std::endl;
        std::cout << "  recoveryPeriod:     " << RECOVERY_PERIOD << " seconds"
                  << std::endl << std::endl;
        std::cout << "A PromQL alarm starts in the OK state rather than "
                     "INSUFFICIENT_DATA,"
                  << std::endl
                  << "which is another way it differs from a classic alarm." <<
std::endl;

        return true;
    }

    //! Report the alarm's contributors, one per series the query matched.
    /*!
      \param client: A CloudWatch client.
      \param alarmName: The name of the alarm.
      \return bool: Function succeeded.
     */
    bool inspectAlarmContributors(const Aws::CloudWatch::CloudWatchClient
&client,
                                  const Aws::String &alarmName) {
        std::cout << "5. Inspect the alarm's contributors" << std::endl <<
std::endl;
        std::cout << "Each contributor is one series the query matched,
identified by its"
                  << std::endl
                  << "label set. This is how you find out which host is unhealthy
rather"
                  << std::endl
```

```
                  << "than only that something is. Classic alarms have no
equivalent."
                  << std::endl << std::endl;

        Aws::Vector<Aws::CloudWatch::Model::AlarmContributor> contributors;
        Aws::CloudWatch::Model::DescribeAlarmContributorsRequest request;
        request.SetAlarmName(alarmName);

        bool done = false;
        while (!done) {
            auto outcome = client.DescribeAlarmContributors(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to describe alarm contributors: "
                          << outcome.GetError().GetMessage() << std::endl;
                return false;
            }

            const auto &page = outcome.GetResult().GetAlarmContributors();
            contributors.insert(contributors.end(), page.begin(), page.end());

            const auto &nextToken = outcome.GetResult().GetNextToken();
            request.SetNextToken(nextToken);
            // A page can come back empty while still carrying a token, so keep
going
            // until the token itself is gone rather than stopping at the first
empty
            // page.
            done = nextToken.empty();
        }

        if (contributors.empty()) {
            std::cout << "No contributors yet. The query matched no series, which
usually"
                      << std::endl
                      << "means no OTel metrics with these labels have arrived.
Once your"
                      << std::endl
                      << "collector is sending data, each matching series appears
here"
                      << std::endl
                      << "with its labels and the reason it breached." <<
std::endl;
            return true;
        }
```

```
        std::cout << "Found " << contributors.size() << " contributors:" <<
std::endl;
        for (const auto &contributor : contributors) {
            std::cout << "  " << contributor.GetContributorId() << ": ";
            bool first = true;
            for (const auto &label : contributor.GetContributorAttributes()) {
                if (!first) {
                    std::cout << ", ";
                }
                std::cout << label.first << "=" << label.second;
                first = false;
            }
            std::cout << std::endl;
            std::cout << "    reason: " << contributor.GetStateReason() <<
std::endl;
        }

        return true;
    }

    //! Build a single-widget dashboard body that charts the given metric.
    /*!
      \param metric: The metric to chart.
      \param region: The region the metric is in. A metric widget must name its
       region, because a dashboard can chart metrics from several.
      \return Aws::String: The dashboard body, as JSON.
     */
    Aws::String buildDashboardBody(const Aws::CloudWatch::Model::Metric &metric,
                                   const Aws::String &region) {
        const auto &dimensions = metric.GetDimensions();

        // A metric is specified in a widget as a flat array,
        // [namespace, metricName, dimensionName, dimensionValue, ...].
        Aws::Utils::Array<Aws::Utils::Json::JsonValue> metricSpec(
            2 + 2 * dimensions.size());
        metricSpec[0].AsString(metric.GetNamespace());
        metricSpec[1].AsString(metric.GetMetricName());
        size_t index = 2;
        for (const auto &dimension : dimensions) {
            metricSpec[index++].AsString(dimension.GetName());
            metricSpec[index++].AsString(dimension.GetValue());
        }
```

```
        Aws::Utils::Array<Aws::Utils::Json::JsonValue> metricsArray(1);
        metricsArray[0].AsArray(metricSpec);

        Aws::Utils::Json::JsonValue textProperties;
        textProperties.WithString(
            "markdown",
            "This dashboard was created programmatically by an AWS SDK code
example.");

        Aws::Utils::Json::JsonValue textWidget;
        textWidget.WithString("type", "text")
            .WithInteger("x", 0)
            .WithInteger("y", 0)
            .WithInteger("width", 24)
            .WithInteger("height", 2)
            .WithObject("properties", textProperties);

        Aws::Utils::Json::JsonValue metricProperties;
        metricProperties.WithArray("metrics", metricsArray)
            .WithString("view", "timeSeries")
            .WithString("stat", "Average")
            .WithInteger("period", 300)
            .WithString("region", region)
            .WithString("title", metric.GetMetricName());

        Aws::Utils::Json::JsonValue metricWidget;
        metricWidget.WithString("type", "metric")
            .WithInteger("x", 0)
            .WithInteger("y", 2)
            .WithInteger("width", 12)
            .WithInteger("height", 6)
            .WithObject("properties", metricProperties);

        Aws::Utils::Array<Aws::Utils::Json::JsonValue> widgets(2);
        widgets[0] = textWidget;
        widgets[1] = metricWidget;

        Aws::Utils::Json::JsonValue body;
        body.WithArray("widgets", widgets);

        return body.View().WriteCompact();
    }

    //! Get statistics for a metric and chart it on a dashboard.
```

```
   /*!
     \param client: A CloudWatch client.
     \param metric: The metric to chart. If its name is empty, this step is
skipped.
     \param dashboardName: The name of the dashboard to create.
     \param dashboardCreated: Set to true if a dashboard was created, so that
cleanup
            only deletes a dashboard that exists.
     \return bool: Function succeeded.
    */
   bool getStatisticsAndChartMetric(const Aws::CloudWatch::CloudWatchClient
&client,
                                    const Aws::CloudWatch::Model::Metric
&metric,
                                    const Aws::String &dashboardName,
                                    const Aws::String &region,
                                    bool &dashboardCreated) {
       std::cout << "6. Get statistics and chart the metric on a dashboard" <<
std::endl
                 << std::endl;
       std::cout << "Statistics and dashboards are how you see what the alarm
is"
                 << std::endl << "evaluating." << std::endl << std::endl;

       if (metric.GetMetricName().empty()) {
           std::cout << "Skipping statistics and dashboard because no metrics
exist yet."
                     << std::endl;
           return true;
       }

       const auto now = std::chrono::system_clock::now();
       Aws::CloudWatch::Model::GetMetricStatisticsRequest statsRequest;
       statsRequest.SetNamespace(metric.GetNamespace());
       statsRequest.SetMetricName(metric.GetMetricName());
       statsRequest.SetDimensions(metric.GetDimensions());
       statsRequest.SetStartTime(Aws::Utils::DateTime(now -
std::chrono::hours(24)));
       statsRequest.SetEndTime(Aws::Utils::DateTime(now));
       statsRequest.SetPeriod(3600);
       statsRequest.AddStatistics(Aws::CloudWatch::Model::Statistic::Average);
       statsRequest.AddStatistics(Aws::CloudWatch::Model::Statistic::Maximum);

       auto statsOutcome = client.GetMetricStatistics(statsRequest);
```

```
        if (!statsOutcome.IsSuccess()) {
            std::cerr << "Failed to get metric statistics: "
                      << statsOutcome.GetError().GetMessage() << std::endl;
            return false;
        }

        const auto &datapoints = statsOutcome.GetResult().GetDatapoints();
        std::cout << "Statistics for " << metric.GetNamespace() << " "
                  << metric.GetMetricName() << " over the last day:" <<
std::endl;
        std::cout << "  Datapoints: " << datapoints.size() << std::endl;
        size_t shown = 0;
        for (const auto &datapoint : datapoints) {
            if (shown++ >= 3) {
                break;
            }
            std::cout << "  " << datapoint.GetTimestamp().ToGmtString(
                                     Aws::Utils::DateFormat::ISO_8601)
                      << " average " << datapoint.GetAverage() << ", maximum "
                      << datapoint.GetMaximum() << std::endl;
        }

        Aws::CloudWatch::Model::PutDashboardRequest putRequest;
        putRequest.SetDashboardName(dashboardName);
        putRequest.SetDashboardBody(buildDashboardBody(metric, region));

        auto putOutcome = client.PutDashboard(putRequest);
        if (!putOutcome.IsSuccess()) {
            std::cerr << "Failed to put dashboard: " <<
putOutcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
        dashboardCreated = true;

        for (const auto &message :
             putOutcome.GetResult().GetDashboardValidationMessages()) {
            std::cout << "Dashboard validation message: " << message.GetMessage()
                      << std::endl;
        }
        std::cout << "Created dashboard " << dashboardName << "." << std::endl;

        Aws::CloudWatch::Model::GetDashboardRequest getRequest;
        getRequest.SetDashboardName(dashboardName);
```

```
        auto getOutcome = client.GetDashboard(getRequest);
        if (getOutcome.IsSuccess()) {
            std::cout << "Read the dashboard back, "
                      << getOutcome.GetResult().GetDashboardBody().size()
                      << " characters of widget JSON." << std::endl;
        }

        return true;
    }

    //! Mute the alarm for a recurring maintenance window.
    /*!
      \param client: A CloudWatch client.
      \param muteRuleName: The name of the mute rule to create.
      \param alarmName: The alarm to target.
      \return bool: Function succeeded.
     */
    bool muteAlarmForMaintenance(const Aws::CloudWatch::CloudWatchClient &client,
                                 const Aws::String &muteRuleName,
                                 const Aws::String &alarmName) {
        std::cout << "7. Mute the alarm for a maintenance window" << std::endl
                  << std::endl;
        std::cout << "While a mute rule is active the targeted alarms keep
evaluating and"
                  << std::endl
                  << "keep changing state, but their actions do not fire. This is
the"
                  << std::endl
                  << "supported way to suppress notifications during planned
maintenance,"
                  << std::endl
                  << "instead of disabling alarm actions and hoping someone
remembers to"
                  << std::endl
                  << "turn them back on." << std::endl << std::endl;

        // The expression is a five-field cron expression,
        // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
five
        // fields, not the six that Amazon EventBridge uses. For a one-time
window, use
        // at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
duration
        // from PT1M to P15D, so PT2H rather than 2h.
```

```
        const Aws::String expression("cron(0 2 * * SUN)");
        const Aws::String duration("PT2H");
        const Aws::String timezone("America/Los_Angeles");

        Aws::CloudWatch::Model::Schedule schedule;
        schedule.SetExpression(expression);
        schedule.SetDuration(duration);
        schedule.SetTimezone(timezone);

        Aws::CloudWatch::Model::Rule rule;
        rule.SetSchedule(schedule);

        // Target up to 100 alarms. If MuteTargets is not set, the rule applies
to every
        // alarm in the account.
        Aws::CloudWatch::Model::MuteTargets muteTargets;
        muteTargets.AddAlarmNames(alarmName);

        Aws::CloudWatch::Model::PutAlarmMuteRuleRequest putRequest;
        putRequest.SetName(muteRuleName);
        putRequest.SetDescription(
            "A mute rule created by the AWS SDK for C++ Basics scenario.");
        putRequest.SetRule(rule);
        putRequest.SetMuteTargets(muteTargets);

        auto putOutcome = client.PutAlarmMuteRule(putRequest);
        if (!putOutcome.IsSuccess()) {
            std::cerr << "Failed to put alarm mute rule: "
                      << putOutcome.GetError().GetMessage() << std::endl;
            return false;
        }

        std::cout << "Created mute rule " << muteRuleName << ":" << std::endl;
        std::cout << "  schedule: " << expression << " for " << duration <<
std::endl;
        std::cout << "  timezone: " << timezone << std::endl;
        std::cout << "  targets:  " << alarmName << std::endl << std::endl;
        std::cout << "Note the two formats here. The expression is a five-field
cron"
                  << std::endl
                  << "expression, five rather than the six Amazon EventBridge
uses. The"
                  << std::endl
```

```
                  << "duration is an ISO 8601 duration, so \"PT2H\" and not \"2h
\"."
                  << std::endl << std::endl;
        std::cout << "Also note that MuteTargets is set explicitly. If you leave
 it out,"
                  << std::endl
                  << "the rule applies to every alarm in the account." <<
 std::endl;

        Aws::CloudWatch::Model::GetAlarmMuteRuleRequest getRequest;
        getRequest.SetAlarmMuteRuleName(muteRuleName);
        auto getOutcome = client.GetAlarmMuteRule(getRequest);
        if (getOutcome.IsSuccess()) {
            const auto &result = getOutcome.GetResult();
            std::cout << "Read the rule back: status "
                      << Aws::CloudWatch::Model::AlarmMuteRuleStatusMapper::
                             GetNameForAlarmMuteRuleStatus(result.GetStatus())
                      << ", mute type " << result.GetMuteType() << "." <<
 std::endl;
        }

        Aws::Vector<Aws::CloudWatch::Model::AlarmMuteRuleSummary> summaries;
        Aws::CloudWatch::Model::ListAlarmMuteRulesRequest listRequest;
        listRequest.SetAlarmName(alarmName);
        bool done = false;
        while (!done) {
            auto listOutcome = client.ListAlarmMuteRules(listRequest);
            if (!listOutcome.IsSuccess()) {
                std::cerr << "Failed to list alarm mute rules: "
                          << listOutcome.GetError().GetMessage() << std::endl;
                return false;
            }

            const auto &page =
 listOutcome.GetResult().GetAlarmMuteRuleSummaries();
            summaries.insert(summaries.end(), page.begin(), page.end());

            const auto &nextToken = listOutcome.GetResult().GetNextToken();
            listRequest.SetNextToken(nextToken);
            done = nextToken.empty();
        }

        std::cout << "Found " << summaries.size()
                  << " mute rules targeting this alarm." << std::endl;
```

```
        // Mute rule summaries carry no name field, only an ARN, so match on the
ARN
        // suffix.
        for (const auto &summary : summaries) {
            const auto &arn = summary.GetAlarmMuteRuleArn();
            const Aws::String slashSuffix = "/" + muteRuleName;
            const Aws::String colonSuffix = ":" + muteRuleName;
            if ((arn.size() >= slashSuffix.size() &&
                 arn.compare(arn.size() - slashSuffix.size(), slashSuffix.size(),
                             slashSuffix) == 0) ||
                (arn.size() >= colonSuffix.size() &&
                 arn.compare(arn.size() - colonSuffix.size(), colonSuffix.size(),
                             colonSuffix) == 0)) {
                std::cout << "  matched by ARN: " << arn << " ("
                          << Aws::CloudWatch::Model::AlarmMuteRuleStatusMapper::

GetNameForAlarmMuteRuleStatus(summary.GetStatus())
                          << ")" << std::endl;
            }
        }

        return true;
    }

    //! Delete everything the scenario created.
    /*!
      \param client: A CloudWatch client.
      \param alarmName: The alarm to delete.
      \param dashboardName: The dashboard to delete.
      \param muteRuleName: The mute rule to delete.
      \param dashboardCreated: Whether a dashboard was created.
      \param startedEnrichment: Whether this run started OTel enrichment.
      \return bool: Every deletion succeeded.
     */
    bool cleanUp(const Aws::CloudWatch::CloudWatchClient &client,
                 const Aws::String &alarmName, const Aws::String &dashboardName,
                 const Aws::String &muteRuleName, bool dashboardCreated,
                 bool startedEnrichment) {
        std::cout << "8. Clean up" << std::endl << std::endl;

        // Each deletion is attempted independently so that one failure does not
leave
        // the remaining resources behind.
        bool result = true;
```

```
        Aws::CloudWatch::Model::DeleteAlarmMuteRuleRequest muteRuleRequest;
        muteRuleRequest.SetAlarmMuteRuleName(muteRuleName);
        auto muteRuleOutcome = client.DeleteAlarmMuteRule(muteRuleRequest);
        if (muteRuleOutcome.IsSuccess()) {
            std::cout << "Deleted mute rule " << muteRuleName << "." <<
std::endl;
        } else {
            std::cerr << "Could not delete the mute rule: "
                      << muteRuleOutcome.GetError().GetMessage() << std::endl;
            result = false;
        }

        Aws::CloudWatch::Model::DeleteAlarmsRequest alarmRequest;
        alarmRequest.AddAlarmNames(alarmName);
        auto alarmOutcome = client.DeleteAlarms(alarmRequest);
        if (alarmOutcome.IsSuccess()) {
            std::cout << "Deleted alarm " << alarmName << "." << std::endl;
        } else {
            std::cerr << "Could not delete the alarm: "
                      << alarmOutcome.GetError().GetMessage() << std::endl;
            result = false;
        }

        if (dashboardCreated) {
            Aws::CloudWatch::Model::DeleteDashboardsRequest dashboardRequest;
            dashboardRequest.AddDashboardNames(dashboardName);
            auto dashboardOutcome = client.DeleteDashboards(dashboardRequest);
            if (dashboardOutcome.IsSuccess()) {
                std::cout << "Deleted dashboard " << dashboardName << "." <<
std::endl;
            } else {
                std::cerr << "Could not delete the dashboard: "
                          << dashboardOutcome.GetError().GetMessage() <<
std::endl;
                result = false;
            }
        }

        if (!startedEnrichment) {
            std::cout << "Left OTel enrichment running, because it was already on
before"
                      << std::endl << "this run." << std::endl;
            return result;
```

```
        }

        Aws::CloudWatch::Model::StopOTelEnrichmentRequest stopRequest;
        auto stopOutcome = client.StopOTelEnrichment(stopRequest);
        if (stopOutcome.IsSuccess()) {
            std::cout << "Stopped OTel enrichment, because this run started it."
                      << std::endl;
        } else {
            std::cerr << "Could not stop OTel enrichment: "
                      << stopOutcome.GetError().GetMessage() << std::endl;
            result = false;
        }

        return result;
    }
} // namespace

//! Run the Amazon CloudWatch Basics scenario.
/*!
  \param query: The PromQL query to alarm on.
  \param clientConfig: AWS client configuration.
  \return bool: Function succeeded.
 */
bool runCloudWatchScenario(const Aws::String &query,
                           const Aws::Client::ClientConfiguration &clientConfig)
 {
    Aws::CloudWatch::CloudWatchClient client(clientConfig);

    // Suffix the resource names so repeated runs do not collide.
    std::mt19937 generator(std::random_device{}());
    const int suffix = std::uniform_int_distribution<int>(1000, 9999)(generator);
    const Aws::String alarmName =
        "doc-example-promql-alarm-" + std::to_string(suffix);
    const Aws::String dashboardName = "doc-example-dashboard-" +
 std::to_string(suffix);
    const Aws::String muteRuleName = "doc-example-mute-rule-" +
 std::to_string(suffix);

    Aws::CloudWatch::Model::Metric metric;
    bool startedEnrichment = false;
    bool dashboardCreated = false;
    bool result = true;

    std::cout << DASHES << std::endl;
```

```
   std::cout << "Welcome to the Amazon CloudWatch Basics scenario." << std::endl
             << std::endl;
   std::cout << "CloudWatch now ingests OpenTelemetry metrics natively. This
scenario"
             << std::endl
             << "walks through that experience: it turns on OTel enrichment so"
             << std::endl
             << "CloudWatch can correlate incoming OTLP metrics with the
resources that"
             << std::endl
             << "produced them, alarms on those metrics with a PromQL query, and
shows"
             << std::endl
             << "you which individual series drove the alarm." << std::endl <<
std::endl;
   std::cout << "A PromQL alarm works differently from a classic metric alarm.
Rather"
             << std::endl
             << "than watching one metric and counting breaching periods, it
evaluates"
             << std::endl
             << "a query that can match many series at once, and tracks each
one"
             << std::endl << "separately as a contributor." << std::endl;
   std::cout << DASHES << std::endl;
   pressEnter();

   // Every step prompts for Enter before the next one, but only while the
scenario is
   // still healthy. Once a step fails the remaining steps are skipped, so there
is
   // nothing left to wait for and the run goes straight to cleanup.
   if (!listMetricsAndNamespaces(client, metric)) {
       result = false;
   }
   std::cout << DASHES << std::endl;
   if (result) {
       pressEnter();
   }

   if (result && !startOTelEnrichment(client, startedEnrichment)) {
       result = false;
   }
   std::cout << DASHES << std::endl;
```

```
    if (result) {
        pressEnter();
    }

    if (result) {
        explainOtlpIngestion();
        std::cout << DASHES << std::endl;
        pressEnter();
    }

    if (result && !createPromQLAlarm(client, alarmName, query)) {
        result = false;
    }
    std::cout << DASHES << std::endl;
    if (result) {
        pressEnter();
    }

    if (result && !inspectAlarmContributors(client, alarmName)) {
        result = false;
    }
    std::cout << DASHES << std::endl;
    if (result) {
        pressEnter();
    }

    if (result && !getStatisticsAndChartMetric(client, metric, dashboardName,
                                               clientConfig.region,
                                               dashboardCreated)) {
        result = false;
    }
    std::cout << DASHES << std::endl;
    if (result) {
        pressEnter();
    }

    if (result && !muteAlarmForMaintenance(client, muteRuleName, alarmName)) {
        result = false;
    }
    std::cout << DASHES << std::endl;
    if (result) {
        pressEnter();
    }
```

```
    // Clean up regardless of whether an earlier step failed, so a partial run
 does not
    // leave resources behind.
    if (!cleanUp(client, alarmName, dashboardName, muteRuleName,
 dashboardCreated,
                 startedEnrichment)) {
        result = false;
    }
    std::cout << DASHES << std::endl;

    std::cout << "This concludes the Amazon CloudWatch Basics scenario." <<
 std::endl;
    std::cout << DASHES << std::endl;

    return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetDashboard
  - GetMetricStatistics
  - GetOTelEnrichment
  - ListAlarmMuteRules
  - ListDashboards
  - ListMetrics
  - PutAlarmMuteRule
  - PutDashboard
  - PutMetricAlarm
  - StartOTelEnrichment
  - StopOTelEnrichment

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating the CloudWatch OpenTelemetry experience.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudwatch.model.AlarmContributor;
import software.amazon.awssdk.services.cloudwatch.model.AlarmMuteRuleSummary;
import software.amazon.awssdk.services.cloudwatch.model.Dimension;
import software.amazon.awssdk.services.cloudwatch.model.GetAlarmMuteRuleResponse;

import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This scenario demonstrates the Amazon CloudWatch OpenTelemetry (OTel)
 experience.
 * CloudWatch ingests OpenTelemetry metrics natively, and this example walks
 through what
 * you do with them: turning on enrichment so CloudWatch can correlate incoming
 OTLP
```

```
 * metrics with the resources that produced them, alarming on those metrics with
 a PromQL
 * query, and finding out which individual series drove the alarm.
 *
 * A PromQL alarm works differently from a classic metric alarm. Rather than
 watching one
 * metric and counting breaching periods, it evaluates a query that can match
 many series
 * at once, and tracks each matching series separately as a contributor.
 *
 * Note that sending OTLP metrics to CloudWatch is not an AWS SDK operation.
 Metrics
 * arrive over the OTLP protocol through the CloudWatch agent, an OpenTelemetry
 * Collector, or an ADOT SDK. Everything this scenario does is configuration and
 querying
 * around that ingestion path.
 *
 * This Java code example performs the following tasks:
 *
 * 1. List metrics and namespaces from Amazon CloudWatch.
 * 2. Start OpenTelemetry enrichment for the account.
 * 3. Explain how OTLP metrics reach CloudWatch.
 * 4. Create an alarm that evaluates a PromQL query.
 * 5. Inspect the contributors to the PromQL alarm.
 * 6. Get metric statistics and chart the metric on a dashboard.
 * 7. Mute the alarm for a maintenance window.
 * 8. Clean up the Amazon CloudWatch resources.
 */
public class CloudWatchScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    private static final String DEFAULT_QUERY = "avg by (host)
 (system_cpu_utilization) > 80";

    // Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to
 3600 seconds.
    private static final int EVALUATION_INTERVAL = 60;
    private static final int PENDING_PERIOD = 300;
    private static final int RECOVERY_PERIOD = 120;

    static CloudWatchActions cwActions = new CloudWatchActions();
```

```
   private static final Logger logger =
LoggerFactory.getLogger(CloudWatchScenario.class);
   static Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) throws Throwable {

       // Suffix the resource names so repeated runs do not collide.
       String suffix = String.valueOf(new Random().nextInt(9000) + 1000);
       String alarmName = "doc-example-promql-alarm-" + suffix;
       String dashboardName = "doc-example-dashboard-" + suffix;
       String muteRuleName = "doc-example-mute-rule-" + suffix;

       logger.info(DASHES);
       logger.info("Welcome to the Amazon CloudWatch Basics scenario.");
       logger.info("""
           CloudWatch now ingests OpenTelemetry metrics natively. This scenario
walks through
           that experience: it turns on OTel enrichment so CloudWatch can
correlate incoming
           OTLP metrics with the resources that produced them, alarms on those
metrics with a
           PromQL query, and shows you which individual series drove the alarm.

           A PromQL alarm works differently from a classic metric alarm. Rather
than watching
           one metric and counting breaching periods, it evaluates a query that
can match many
           series at once, and tracks each one separately as a contributor.

           Let's get started...
           """);
       waitForInputToContinue(scanner);

       try {
           runScenario(alarmName, dashboardName, muteRuleName);
       } catch (RuntimeException e) {
           e.printStackTrace();
       }
       logger.info(DASHES);
   }

   private static void runScenario(String alarmName, String dashboardName,
String muteRuleName)
           throws Throwable {
```

```
        // Tracks whether this run turned enrichment on, so that cleanup only
turns off
        // enrichment that this run started.
        boolean startedEnrichment = false;

        logger.info(DASHES);
        logger.info("""
            1. List metrics and namespaces

            Before configuring anything, let's see what CloudWatch is already
collecting in
            this account by calling ListMetrics.
            """);
        waitForInputToContinue(scanner);

        ArrayList<String> namespaces = cwActions.listNameSpacesAsync().join();
        logger.info("Found {} namespaces in this account:", namespaces.size());
        namespaces.stream().limit(10).forEach(namespace -> logger.info("  {}",
namespace));
        if (namespaces.isEmpty()) {
            logger.info("""
                No metrics found in this account. The statistics and dashboard
steps later on
                need an existing metric, so they will be skipped.
                """);
        }
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("""
            2. Start OpenTelemetry enrichment

            Enrichment is what lets CloudWatch attach AWS resource context to the
OTLP metrics
            you send it. Without it, your metrics arrive as opaque series with no
connection to
            the resources that emitted them.

            We check the current state first, and only start enrichment if it
isn't already on.
            """);
        waitForInputToContinue(scanner);
```

```
        String status = cwActions.getOTelEnrichmentStatusAsync().join();
        logger.info("Enrichment status: {}", status);

        if (!"Running".equalsIgnoreCase(status)) {
            cwActions.startOTelEnrichmentAsync().join();
            startedEnrichment = true;
            status = cwActions.getOTelEnrichmentStatusAsync().join();
            logger.info("Enrichment status: {}", status);
            logger.info("""
                Note: this run started enrichment, so the cleanup step will stop
it again.
                """);
        } else {
            logger.info("""
                Enrichment was already running, so we will leave it alone. The
cleanup step
                will not stop it, because other workloads in this account may
depend on it.
                """);
        }
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("""
            3. Send OTLP metrics to CloudWatch

            This step is not an AWS SDK operation, and that's worth being
explicit about.
            Metrics reach CloudWatch over the OTLP protocol, through the
CloudWatch agent, an
            OpenTelemetry Collector, or an ADOT SDK. There is no PutOTelMetrics
API to call.

            Point your collector at the CloudWatch metrics endpoint, which
follows the pattern
            https://monitoring.<region>.amazonaws.com/v1/metrics

            The endpoint is HTTP/1.1 only and does not support gRPC, so use an
otlphttp
            exporter rather than otlp. The metrics endpoint signs as
"monitoring".
            """);
        waitForInputToContinue(scanner);
```

```
        logger.info(DASHES);
        logger.info("""
            4. Create a PromQL alarm

            Now we alarm on those metrics. The comparison goes inside the query
itself: a
            PromQL alarm has no separate threshold, comparison operator,
statistic, or period.
            """);
        logger.info("Enter a PromQL query, or press <ENTER> for the default");
        logger.info("[{}]:", DEFAULT_QUERY);
        String queryInput = scanner.nextLine();
        String query = queryInput == null || queryInput.isBlank() ?
DEFAULT_QUERY : queryInput.trim();

        cwActions.putPromQLMetricAlarmAsync(alarmName, query,
EVALUATION_INTERVAL,
                PENDING_PERIOD, RECOVERY_PERIOD).join();
        logger.info("Created alarm {}:", alarmName);
        logger.info("  query:              {}", query);
        logger.info("  evaluationInterval: {} seconds", EVALUATION_INTERVAL);
        logger.info("  pendingPeriod:      {} seconds", PENDING_PERIOD);
        logger.info("  recoveryPeriod:     {} seconds", RECOVERY_PERIOD);
        logger.info("""

            A PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA,
which is
            another way it differs from a classic alarm.
            """);
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("""
            5. Inspect the alarm's contributors

            Each contributor is one series the query matched, identified by its
label set. This
            is how you find out which host is unhealthy rather than only that
something is.
            Classic alarms have no equivalent.
            """);
        waitForInputToContinue(scanner);
```

```
        List<AlarmContributor> contributors =
cwActions.describeAlarmContributorsAsync(alarmName).join();
        if (contributors.isEmpty()) {
            logger.info("""
                No contributors yet. The query matched no series, which usually
means no OTel
                metrics with these labels have arrived. Once your collector is
sending data,
                each matching series appears here with its labels and the reason
it breached.
                """);
        } else {
            logger.info("Found {} contributors:", contributors.size());
            for (AlarmContributor contributor : contributors) {
                StringBuilder labels = new StringBuilder();
                for (Map.Entry<String, String> attribute :
contributor.contributorAttributes().entrySet()) {
                    if (labels.length() > 0) {
                        labels.append(", ");
                    }

labels.append(attribute.getKey()).append("=").append(attribute.getValue());
                }
                logger.info("  {}: {}", contributor.contributorId(), labels);
                logger.info("    reason: {}", contributor.stateReason());
            }
        }
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("""
            6. Get statistics and chart the metric on a dashboard

            Statistics and dashboards are how you see what the alarm is
evaluating.
            """);
        waitForInputToContinue(scanner);

        boolean dashboardCreated = false;
        if (!namespaces.isEmpty()) {
            String namespace = namespaces.get(0);
            ArrayList<String> metrics =
cwActions.listMetsAsync(namespace).join();
            if (metrics != null && !metrics.isEmpty()) {
```

```
                String metricName = metrics.get(0);
                String startDate = Instant.now().minus(24,
ChronoUnit.HOURS).toString();
                Dimension dimension = null;
                try {
                    dimension = cwActions.getSpecificMetAsync(namespace).join();
                    cwActions.getAndDisplayMetricStatisticsAsync(namespace,
metricName,
                            "Average", startDate, dimension).join();
                } catch (RuntimeException e) {
                    logger.info("Could not get statistics for {}/{}: {}",
namespace, metricName,
                            e.getMessage());
                }

                // Chart the metric this run just discovered. Reading the widgets
from a
                // file would chart metrics that may not exist in this account.
                try {
                    String dashboardBody = buildDashboardBody(namespace,
metricName, dimension,
                            cwActions.getRegion());
                    cwActions.createDashboardAsync(dashboardName,
dashboardBody).join();
                    dashboardCreated = true;
                    cwActions.listDashboardsAsync().join();
                } catch (RuntimeException e) {
                    logger.info("Could not create the dashboard: {}",
e.getMessage());
                }
            } else {
                logger.info("No metrics found in namespace {}, skipping
statistics and the "
                        + "dashboard.", namespace);
            }
        } else {
            logger.info("Skipping statistics and dashboard because no metrics
exist yet.");
        }
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("""
            7. Mute the alarm for a maintenance window
```

```
            While a mute rule is active the targeted alarms keep evaluating and
keep changing
            state, but their actions do not fire. This is the supported way to
suppress
            notifications during planned maintenance, instead of disabling alarm
actions and
            hoping someone remembers to turn them back on.
            """);
        waitForInputToContinue(scanner);

        // The expression is a five-field cron expression,
        // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
five fields,
        // not the six that Amazon EventBridge uses. For a one-time window, use
        // at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
duration from
        // PT1M to P15D, so PT2H rather than 2h.
        String expression = "cron(0 2 * * SUN)";
        String duration = "PT2H";
        String timezone = "America/Los_Angeles";

        cwActions.putAlarmMuteRuleAsync(muteRuleName, expression, duration,
timezone,
                List.of(alarmName)).join();
        logger.info("Created mute rule {}:", muteRuleName);
        logger.info("  schedule: {} for {}", expression, duration);
        logger.info("  timezone: {}", timezone);
        logger.info("  targets:  {}", alarmName);
        logger.info("""

            Note the two formats here. The expression is a five-field cron
expression, five
            rather than the six Amazon EventBridge uses. The duration is an ISO
8601 duration,
            so 'PT2H' and not '2h'.

            Also note that muteTargets is set explicitly. If you leave it out,
the rule applies
            to every alarm in the account.
            """);

        GetAlarmMuteRuleResponse muteRule =
cwActions.getAlarmMuteRuleAsync(muteRuleName).join();
```

```
        logger.info("Read the rule back: status {}, mute type {}.",
muteRule.statusAsString(),
                muteRule.muteType());

        List<AlarmMuteRuleSummary> summaries =
cwActions.listAlarmMuteRulesAsync(alarmName).join();
        logger.info("Found {} mute rules targeting this alarm.",
summaries.size());
        // Mute rule summaries carry no name field, only an ARN, so match on the
ARN suffix.
        summaries.stream()
                .filter(summary -> summary.alarmMuteRuleArn().endsWith("/" +
muteRuleName)
                        || summary.alarmMuteRuleArn().endsWith(":" +
muteRuleName))
                .findFirst()
                .ifPresent(summary -> logger.info("  matched by ARN: {} ({})",
                        summary.alarmMuteRuleArn(), summary.statusAsString()));
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("8. Clean up");
        logger.info("Delete the resources this scenario created? (y/n)");
        String cleanUp = scanner.nextLine();
        if (cleanUp == null || !cleanUp.trim().equalsIgnoreCase("y")) {
            logger.info("""
                Skipping cleanup. Note that the alarm, dashboard, and mute rule
are still in
                your account, and enrichment may still be running.
                """);
            logger.info(DASHES);
            logger.info("This concludes the Amazon CloudWatch Basics scenario.");
            return;
        }

        // Each deletion is attempted independently so that one failure does not
leave the
        // remaining resources behind.
        try {
            cwActions.deleteAlarmMuteRuleAsync(muteRuleName).join();
        } catch (RuntimeException e) {
            logger.info("Could not delete the mute rule: {}", e.getMessage());
        }
```

```
        try {
            cwActions.deleteCWAlarmAsync(alarmName).join();
            logger.info("Deleted alarm {}.", alarmName);
        } catch (RuntimeException e) {
            logger.info("Could not delete the alarm: {}", e.getMessage());
        }

        if (dashboardCreated) {
            try {
                cwActions.deleteDashboardAsync(dashboardName).join();
                logger.info("Deleted dashboard {}.", dashboardName);
            } catch (RuntimeException e) {
                logger.info("Could not delete the dashboard: {}",
e.getMessage());
            }
        }

        if (startedEnrichment) {
            try {
                cwActions.stopOTelEnrichmentAsync().join();
                logger.info("Stopped OTel enrichment, because this run started
it.");
            } catch (RuntimeException e) {
                logger.info("Could not stop OTel enrichment: {}",
e.getMessage());
            }
        } else {
            logger.info("""
                Left OTel enrichment running, because it was already on before
this run.
                """);
        }

        logger.info(DASHES);
        logger.info("This concludes the Amazon CloudWatch Basics scenario.");
        logger.info(DASHES);
    }

    /**
     * Builds a single-widget dashboard body that charts the given metric.
     *
     * @param metricNamespace the namespace of the metric to chart
     * @param metricName      the name of the metric to chart
```

```
    * @param dimension       a dimension to narrow the metric to, or null for
none
    * @param region          the Region the metric is in. A metric widget must
name its
    *                        Region, because a dashboard can chart metrics from
several.
    * @return the dashboard body, as JSON
    */
   static String buildDashboardBody(String metricNamespace, String metricName,
           Dimension dimension, String region) {
       String dimensionParts = dimension == null ? ""
               : String.format(", \"%s\", \"%s\"", dimension.name(),
dimension.value());

       return String.format("""
           {
               "widgets": [
                   {
                       "type": "text",
                       "x": 0, "y": 0, "width": 24, "height": 2,
                       "properties": {
                           "markdown": "This dashboard was created
programmatically by an AWS SDK code example."
                       }
                   },
                   {
                       "type": "metric",
                       "x": 0, "y": 2, "width": 12, "height": 6,
                       "properties": {
                           "metrics": [[ "%s", "%s"%s ]],
                           "view": "timeSeries",
                           "stat": "Average",
                           "period": 300,
                           "region": "%s",
                           "title": "%s"
                       }
                   }
               ]
           }
           """, metricNamespace, metricName, dimensionParts, region,
metricName);
   }

   private static void waitForInputToContinue(Scanner scanner) {
```

```
        while (true) {
            logger.info("");
            logger.info("Press <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input == null || input.trim().isEmpty()) {
                logger.info("Continuing with the program...");
                logger.info("");
                break;
            } else {
                logger.info("Invalid input. Please try again.");
            }
        }
    }
}
```

A wrapper class for the CloudWatch SDK methods that the scenario calls.

```
public class CloudWatchActions {

    private static CloudWatchAsyncClient cloudWatchAsyncClient;

    private static final Logger logger =
 LoggerFactory.getLogger(CloudWatchActions.class);

    /**
     * Retrieves an asynchronous CloudWatch client instance.
     *
     * <p>
     * This method ensures that the CloudWatch client is initialized with the
 following configurations:
     * <ul>
     *     <li>Maximum concurrency: 100</li>
     *     <li>Connection timeout: 60 seconds</li>
     *     <li>Read timeout: 60 seconds</li>
     *     <li>Write timeout: 60 seconds</li>
     *     <li>API call timeout: 2 minutes</li>
     *     <li>API call attempt timeout: 90 seconds</li>
     *     <li>Retry strategy: STANDARD</li>
     * </ul>
     * </p>
     *
```

```
     * @return the asynchronous CloudWatch client instance
     */
    private static CloudWatchAsyncClient getAsyncClient() {
        if (cloudWatchAsyncClient == null) {
            SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
                .maxConcurrency(100)
                .connectionTimeout(Duration.ofSeconds(60))
                .readTimeout(Duration.ofSeconds(60))
                .writeTimeout(Duration.ofSeconds(60))
                .build();

            ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
                .apiCallTimeout(Duration.ofMinutes(2))
                .apiCallAttemptTimeout(Duration.ofSeconds(90))
                .retryStrategy(RetryMode.STANDARD)
                .build();

            cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return cloudWatchAsyncClient;
    }

    /**
     * Returns the Region the client resolved, which a dashboard's metric widgets
must name.
     *
     * @return the Region ID, such as us-east-1
     */
    public String getRegion() {
        return getAsyncClient().serviceClientConfiguration().region().id();
    }

    /**
     * Deletes an Anomaly Detector.
     *
     * @param fileName the name of the file containing the Anomaly Detector
configuration
     * @return a CompletableFuture that represents the asynchronous deletion of
the Anomaly Detector
     */
```

```
   public CompletableFuture<DeleteAnomalyDetectorResponse>
deleteAnomalyDetectorAsync(String fileName) {
       CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               return new ObjectMapper().readTree(parser); // Return the root
node
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
e);
           }
       });

       return readFileFuture.thenCompose(rootNode -> {
           String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
           String customMetricName =
rootNode.findValue("customMetricName").asText();

           SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
               .metricName(customMetricName)
               .namespace(customMetricNamespace)
               .stat("Maximum")
               .build();

           DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
               .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
               .build();

           return getAsyncClient().deleteAnomalyDetector(request);
       }).whenComplete((result, exception) -> {
           if (exception != null) {
               throw new RuntimeException("Failed to delete the Anomaly
Detector", exception);
           } else {
               logger.info("Successfully deleted the Anomaly Detector.");
           }
       });
   }
```

```
   /**
    * Deletes a CloudWatch alarm.
    *
    * @param alarmName the name of the alarm to be deleted
    * @return a {@link CompletableFuture} representing the asynchronous
operation to delete the alarm
    * the {@link DeleteAlarmsResponse} is returned when the operation completes
successfully,
    * or a {@link RuntimeException} is thrown if the operation fails
    */
   public CompletableFuture<DeleteAlarmsResponse> deleteCWAlarmAsync(String
alarmName) {
       DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
           .alarmNames(alarmName)
           .build();

       return getAsyncClient().deleteAlarms(request)
           .whenComplete((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to delete the alarm:{} " +
alarmName, exception);
               } else {
                   logger.info("Successfully deleted alarm {} ", alarmName);
               }
           });
   }

   /**
    * Deletes the specified dashboard.
    *
    * @param dashboardName the name of the dashboard to be deleted
    * @return a {@link CompletableFuture} representing the asynchronous
operation of deleting the dashboard
    * @throws RuntimeException if the dashboard deletion fails
    */
   public CompletableFuture<DeleteDashboardsResponse>
deleteDashboardAsync(String dashboardName) {
       DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
           .dashboardNames(dashboardName)
           .build();

       return getAsyncClient().deleteDashboards(dashboardsRequest)
           .whenComplete((response, exception) -> {
```

```
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the dashboard: "
+ dashboardName, exception);
                } else {
                    logger.info("{} was successfully deleted.", dashboardName);
                }
            });
    }


    /**
     * Retrieves and saves a custom metric image to a file.
     *
     * @param fileName the name of the file to save the metric image to
     * @return a {@link CompletableFuture} that completes when the image has been
saved to the file
     */
    public CompletableFuture<Void> downloadAndSaveMetricImageAsync(String
fileName) {
        logger.info("Getting Image data for custom metric.");
        String myJSON = """
              {
                  "title": "Example Metric Graph",
                  "view": "timeSeries",
                  "stacked ": false,
                  "period": 10,
                  "width": 1400,
                  "height": 600,
                  "metrics": [
                      [
                      "AWS/Billing",
                      "EstimatedCharges",
                      "Currency",
                      "USD"
                     ]
                  ]
              }
            """;

        GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
            .metricWidget(myJSON)
            .build();
```

```
        return getAsyncClient().getMetricWidgetImage(imageRequest)
            .thenCompose(response -> {
                SdkBytes sdkBytes = response.metricWidgetImage();
                byte[] bytes = sdkBytes.asByteArray();
                return CompletableFuture.runAsync(() -> {
                    try {
                        File outputFile = new File(fileName);
                        try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                            outputStream.write(bytes);
                        }
                    } catch (IOException e) {
                        throw new RuntimeException("Failed to write image to
file", e);
                    }
                });
            })
            .whenComplete((result, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Error getting and saving metric
image", exception);
                } else {
                    logger.info("Image data saved successfully to {}", fileName);
                }
            });
    }


    /**
     * Describes the anomaly detectors based on the specified JSON file.
     *
     * @param fileName the name of the JSON file containing the custom metric
namespace and name
     * @return a {@link CompletableFuture} that completes when the anomaly
detectors have been described
     * @throws RuntimeException if there is a failure during the operation, such
as when reading or parsing the JSON file,
     *                          or when describing the anomaly detectors
     */
    public CompletableFuture<Void> describeAnomalyDetectorsAsync(String fileName)
{
        CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
```

```
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                return new ObjectMapper().readTree(parser);
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            try {
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                    .maxResults(10)
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return
getAsyncClient().describeAnomalyDetectors(detectorsRequest).thenAccept(response
-> {
                    List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
                    for (AnomalyDetector detector : anomalyDetectorList) {
                        logger.info("Metric name: {} ",
detector.singleMetricAnomalyDetector().metricName());
                        logger.info("State: {} ", detector.stateValue());
                    }
                });
            } catch (RuntimeException e) {
                throw new RuntimeException("Failed to describe anomaly
detectors", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error describing anomaly detectors",
exception);
            }
        });
```

```
    }


    /**
     * Adds an anomaly detector for the given file.
     *
     * @param fileName the name of the file containing the anomaly detector
configuration
     * @return a {@link CompletableFuture} that completes when the anomaly
detector has been added
     */
    public CompletableFuture<Void> addAnomalyDetectorAsync(String fileName) {
        CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                return new ObjectMapper().readTree(parser); // Return the root
node
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            try {
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

                PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();
```

```
                return
getAsyncClient().putAnomalyDetector(anomalyDetectorRequest).thenAccept(response
-> {
                    logger.info("Added anomaly detector for metric {}",
customMetricName);
                });
            } catch (Exception e) {
                throw new RuntimeException("Failed to create anomaly detector",
e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error adding anomaly detector",
exception);
            }
        });
    }


    /**
     * Retrieves the alarm history for a given alarm name and date range.
     *
     * @param fileName the path to the JSON file containing the alarm name
     * @param date     the date to start the alarm history search (in the format
"yyyy-MM-dd'T'HH:mm:ss'Z'")
     * @return a {@code CompletableFuture<Void>} that completes when the alarm
history has been retrieved and processed
     */
    public CompletableFuture<Void> getAlarmHistoryAsync(String fileName, String
date) {
        CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
                return rootNode.findValue("exampleAlarmName").asText(); // Return
alarmName from the JSON file
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });
```

```
        // Use the alarm name to describe alarm history with a paginator.
        return readFileFuture.thenCompose(alarmName -> {
            try {
                Instant start = Instant.parse(date);
                Instant endDate = Instant.now();
                DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                    .startDate(start)
                    .endDate(endDate)
                    .alarmName(alarmName)
                    .historyItemType(HistoryItemType.ACTION)
                    .build();

                // Use the paginator to paginate through alarm history pages.
                DescribeAlarmHistoryPublisher historyPublisher =
getAsyncClient().describeAlarmHistoryPaginator(historyRequest);
                CompletableFuture<Void> future = historyPublisher
                    .subscribe(response ->
response.alarmHistoryItems().forEach(item -> {
                        logger.info("History summary: {}",
item.historySummary());
                        logger.info("Timestamp: {}", item.timestamp());
                    }))
                    .whenComplete((result, exception) -> {
                        if (exception != null) {
                            logger.error("Error occurred while getting alarm
history: " + exception.getMessage(), exception);
                        } else {
                            logger.info("Successfully retrieved all alarm
history.");
                        }
                    });

                // Return the future to the calling code for further handling
                return future;
            } catch (Exception e) {
                throw new RuntimeException("Failed to process alarm history", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error completing alarm history
processing", exception);
            }
```

```
        });
    }



    /**
     * Checks for a metric alarm in AWS CloudWatch.
     *
     * @param fileName the name of the file containing the JSON configuration for
the custom metric
     * @return a {@link CompletableFuture} that completes when the check for the
metric alarm is complete
     */
    public CompletableFuture<Void> checkForMetricAlarmAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return checkForAlarmAsync(metricRequest, customMetricName, 10);
```

```
            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error checking metric alarm",
exception);
            }
        });
    }

    // Recursive method to check for the alarm.

    /**
     * Checks for the existence of an alarm asynchronously for the specified
metric.
     *
     * @param metricRequest    the request to describe the alarms for the
specified metric
     * @param customMetricName the name of the custom metric to check for an
alarm
     * @param retries          the number of retries to perform if no alarm is
found
     * @return a {@link CompletableFuture} that completes when an alarm is found
or the maximum number of retries has been reached
     */
    private static CompletableFuture<Void>
checkForAlarmAsync(DescribeAlarmsForMetricRequest metricRequest, String
customMetricName, int retries) {
        if (retries == 0) {
            return CompletableFuture.completedFuture(null).thenRun(() ->
                logger.info("No Alarm state found for {} after 10 retries.",
customMetricName)
            );
        }

        return
(getAsyncClient().describeAlarmsForMetric(metricRequest).thenCompose(response ->
{
            if (response.hasMetricAlarms()) {
                logger.info("Alarm state found for {}", customMetricName);
                return CompletableFuture.completedFuture(null); // Alarm found,
complete the future
```

```
            } else {
                return CompletableFuture.runAsync(() -> {
                    try {
                        Thread.sleep(20000);
                        logger.info(".");
                    } catch (InterruptedException e) {
                        throw new RuntimeException("Interrupted while waiting to
retry", e);
                    }
                }).thenCompose(v -> checkForAlarmAsync(metricRequest,
customMetricName, retries - 1)); // Recursive call
            }
        }));
    }


    /**
     * Adds metric data for an alarm asynchronously.
     *
     * @param fileName the name of the JSON file containing the metric data
     * @return a CompletableFuture that asynchronously returns the
PutMetricDataResponse
     */
    public CompletableFuture<PutMetricDataResponse>
addMetricDataForAlarmAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
```

```
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();
                Instant instant = Instant.now();

                // Create MetricDatum objects.
                MetricDatum datum1 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1001.00)
                    .timestamp(instant)
                    .build();

                MetricDatum datum2 = MetricDatum.builder()
                    .metricName(customMetricName)
                    .unit(StandardUnit.NONE)
                    .value(1002.00)
                    .timestamp(instant)
                    .build();

                List<MetricDatum> metricDataList = new ArrayList<>();
                metricDataList.add(datum1);
                metricDataList.add(datum2);

                // Build the PutMetricData request.
                PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

                // Send the request asynchronously.
                return getAsyncClient().putMetricData(request);

            } catch (IOException e) {
                CompletableFuture<PutMetricDataResponse> failedFuture = new
CompletableFuture<>();
                failedFuture.completeExceptionally(new RuntimeException("Failed
to parse JSON content", e));
                return failedFuture;
            }
        }).whenComplete((response, exception) -> {
            if (exception != null) {
```

```
                logger.error("Failed to put metric data: " +
exception.getMessage(), exception);
            } else {
                logger.info("Added metric values for metric.");
            }
        });
    }


    /**
     * Retrieves custom metric data from the AWS CloudWatch service.
     *
     * @param fileName the name of the file containing the custom metric
information
     * @return a {@link CompletableFuture} that completes when the metric data
has been retrieved
     */
    public CompletableFuture<Void> getCustomMetricDataAsync(String fileName) {
        CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                // Read values from the JSON file.
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
                return rootNode.toString(); // Return JSON as a string for
further processing
            } catch (IOException e) {
                throw new RuntimeException("Failed to read file", e);
            }
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                // Parse the JSON string to extract relevant values.
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                // Set the current time and date range for metric query.
```

```
                Instant nowDate = Instant.now();
                long hours = 1;
                long minutes = 30;
                Instant endTime = nowDate.plus(hours,
ChronoUnit.HOURS).plus(minutes, ChronoUnit.MINUTES);

                Metric met = Metric.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                MetricStat metStat = MetricStat.builder()
                    .stat("Maximum")
                    .period(60)  // Assuming period in seconds
                    .metric(met)
                    .build();

                MetricDataQuery dataQuery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

                List<MetricDataQuery> dq = new ArrayList<>();
                dq.add(dataQuery);

                GetMetricDataRequest getMetricDataRequest =
GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(endTime)
                    .metricDataQueries(dq)
                    .build();

                // Call the async method for CloudWatch data retrieval.
                return getAsyncClient().getMetricData(getMetricDataRequest);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).thenAccept(response -> {
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
```

```
                logger.info("The label is: {}", item.label());
                logger.info("The status code is: {}",
item.statusCode().toString());
            }
        }).exceptionally(exception -> {
            throw new RuntimeException("Failed to get metric data", exception);
        });
    }


    /**
     * Describes the CloudWatch alarms of the 'METRIC_ALARM' type.
     *
     * @return a {@link CompletableFuture} that represents the asynchronous
operation
     * of describing the CloudWatch alarms. The future completes when the
     * operation is finished, either successfully or with an error.
     */
    public CompletableFuture<Void> describeAlarmsAsync() {
        List<AlarmType> typeList = new ArrayList<>();
        typeList.add(AlarmType.METRIC_ALARM);
        DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
            .alarmTypes(typeList)
            .maxRecords(10)
            .build();

        return getAsyncClient().describeAlarms(alarmsRequest)
            .thenAccept(response -> {
                List<MetricAlarm> alarmList = response.metricAlarms();
                for (MetricAlarm alarm : alarmList) {
                    logger.info("Alarm name: {}", alarm.alarmName());
                    logger.info("Alarm description: {} ",
alarm.alarmDescription());
                }
            })
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to describe alarms: {}",
ex.getMessage());
                } else {
                    logger.info("Successfully described alarms.");
                }
            });
    }
```

```
   /**
    * Creates an alarm based on the configuration provided in a JSON file.
    *
    * @param fileName the name of the JSON file containing the alarm
configuration
    * @return a CompletableFuture that represents the asynchronous operation of
creating the alarm
    * @throws RuntimeException if an exception occurs while reading the JSON
file or creating the alarm
    */
   public CompletableFuture<String> createAlarmAsync(String fileName) {
       com.fasterxml.jackson.databind.JsonNode rootNode;
       try {
           JsonParser parser = new JsonFactory().createParser(new
File(fileName));
           rootNode = new ObjectMapper().readTree(parser);
       } catch (IOException e) {
           throw new RuntimeException("Failed to read the alarm configuration
file", e);
       }

       // Extract values from the JSON node.
       String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
       String customMetricName =
rootNode.findValue("customMetricName").asText();
       String alarmName = rootNode.findValue("exampleAlarmName").asText();
       String emailTopic = rootNode.findValue("emailTopic").asText();
       String accountId = rootNode.findValue("accountId").asText();
       String region = rootNode.findValue("region").asText();

       // Create a List for alarm actions.
       List<String> alarmActions = new ArrayList<>();
       alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);

       PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
           .alarmActions(alarmActions)
           .alarmDescription("Example metric alarm")
           .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
           .threshold(100.00)
```

```
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .evaluationPeriods(1)
            .period(10)
            .statistic("Maximum")
            .datapointsToAlarm(1)
            .treatMissingData("ignore")
            .build();

        // Call the putMetricAlarm asynchronously and handle the result.
        return getAsyncClient().putMetricAlarm(alarmRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create alarm: {}", ex.getMessage());
                    throw new RuntimeException("Failed to create alarm", ex);
                } else {
                    logger.info("{} was successfully created!", alarmName);
                    return alarmName;
                }
            });
    }

    /**
     * Adds a metric to a dashboard asynchronously.
     *
     * @param fileName      the name of the file containing the dashboard content
     * @param dashboardName the name of the dashboard to be updated
     * @return a {@link CompletableFuture} representing the asynchronous
operation, which will complete with a
     * {@link PutDashboardResponse} when the dashboard is successfully updated
     */
    public CompletableFuture<PutDashboardResponse>
addMetricToDashboardAsync(String fileName, String dashboardName) {
        String dashboardBody;
        try {
            dashboardBody = readFileAsString(fileName);
        } catch (IOException e) {
            throw new RuntimeException("Failed to read the dashboard file", e);
        }

        PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
            .dashboardName(dashboardName)
            .dashboardBody(dashboardBody)
            .build();
```

```
        return getAsyncClient().putDashboard(dashboardRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to update dashboard: {}",
ex.getMessage());
                    throw new RuntimeException("Error updating dashboard", ex);
                } else {
                    logger.info("{} was successfully updated.", dashboardName);
                    return response;
                }
            });
    }

    /**
     * Creates a new custom metric.
     *
     * @param dataPoint the data point to be added to the custom metric
     * @return a {@link CompletableFuture} representing the asynchronous
operation of adding the custom metric
     */
    public CompletableFuture<PutMetricDataResponse>
createNewCustomMetricAsync(Double dataPoint) {
        Dimension dimension = Dimension.builder()
            .name("UNIQUE_PAGES")
            .value("URLS")
            .build();

        // Set an Instant object for the current time in UTC.
        String time =
ZonedDateTime.now(ZoneOffset.UTC).format(DateTimeFormatter.ISO_INSTANT);
        Instant instant = Instant.parse(time);

        // Create the MetricDatum.
        MetricDatum datum = MetricDatum.builder()
            .metricName("PAGES_VISITED")
            .unit(StandardUnit.NONE)
            .value(dataPoint)
            .timestamp(instant)
            .dimensions(dimension)
            .build();

        PutMetricDataRequest request = PutMetricDataRequest.builder()
            .namespace("SITE/TRAFFIC")
```

```
            .metricData(datum)
            .build();

        return getAsyncClient().putMetricData(request)
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    throw new RuntimeException("Error adding custom metric", ex);
                } else {
                    logger.info("Successfully added metric values for
PAGES_VISITED.");
                }
            });
    }

    /**
     * Lists the available dashboards.
     *
     * @return a {@link CompletableFuture} that completes when the operation is
finished.
     * The future will complete exceptionally if an error occurs while listing
the dashboards.
     */
    public CompletableFuture<Void> listDashboardsAsync() {
        ListDashboardsRequest listDashboardsRequest =
ListDashboardsRequest.builder().build();
        ListDashboardsPublisher paginator =
getAsyncClient().listDashboardsPaginator(listDashboardsRequest);
        return paginator.subscribe(response -> {
            response.dashboardEntries().forEach(entry -> {
                logger.info("Dashboard name is: {} ", entry.dashboardName());
                logger.info("Dashboard ARN is: {} ", entry.dashboardArn());
            });
        }).exceptionally(ex -> {
            logger.info("Failed to list dashboards: {} ", ex.getMessage());
            throw new RuntimeException("Error occurred while listing dashboards",
ex);
        });
    }

    /**
     * Creates a new dashboard with the specified name and the metrics described
by the given file.
     *
     * @param dashboardName the name of the dashboard to be created
```

```
    * @param fileName      the name of the file containing the dashboard body
    * @return a {@link CompletableFuture} representing the asynchronous
operation of creating the dashboard
    * @throws IOException if there is an error reading the dashboard body from
the file
    */
   public CompletableFuture<PutDashboardResponse>
createDashboardWithMetricsAsync(String dashboardName, String fileName) throws
IOException {
       return createDashboardAsync(dashboardName, readFileAsString(fileName));
   }


   /**
    * Creates a new dashboard with the specified name and body.
    *
    * @param dashboardName the name of the dashboard to be created
    * @param dashboardBody the dashboard body, as JSON
    * @return a {@link CompletableFuture} representing the asynchronous
operation of creating the dashboard
    */
   public CompletableFuture<PutDashboardResponse> createDashboardAsync(String
dashboardName, String dashboardBody) {
       PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
           .dashboardName(dashboardName)
           .dashboardBody(dashboardBody)
           .build();

       return getAsyncClient().putDashboard(dashboardRequest)
           .handle((response, ex) -> {
               if (ex != null) {
                   logger.info("Failed to create dashboard: {}",
ex.getMessage());
                   throw new RuntimeException("Dashboard creation failed", ex);
               } else {
                   // Handle the normal response case
                   logger.info("{} was successfully created.", dashboardName);
                   List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
                   if (messages.isEmpty()) {
                       logger.info("There are no messages in the new
Dashboard.");
                   } else {
                       for (DashboardValidationMessage message : messages) {
```

```
                            logger.info("Message: {}", message.message());
                        }
                    }
                    return response; // Return the response for further use
                }
            });
    }


    /**
     * Retrieves the metric statistics for the "EstimatedCharges" metric in the
"AWS/Billing" namespace.
     *
     * @param costDateWeek the start date for the metric statistics, in the
format of an ISO-8601 date string (e.g., "2023-04-05")
     * @return a {@link CompletableFuture} that, when completed, contains the
{@link GetMetricStatisticsResponse} with the retrieved metric statistics
     * @throws RuntimeException if the metric statistics cannot be retrieved
successfully
     */
    public CompletableFuture<GetMetricStatisticsResponse>
getMetricStatisticsAsync(String costDateWeek) {
        Instant start = Instant.parse(costDateWeek);
        Instant endDate = Instant.now();

        // Define dimension
        Dimension dimension = Dimension.builder()
            .name("Currency")
            .value("USD")
            .build();

        List<Dimension> dimensionList = new ArrayList<>();
        dimensionList.add(dimension);

        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .metricName("EstimatedCharges")
            .namespace("AWS/Billing")
            .dimensions(dimensionList)
            .statistics(Statistic.MAXIMUM)
            .startTime(start)
            .endTime(endDate)
            .period(86400) // One day period
            .build();
```

```
        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {})",
datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    throw new RuntimeException("Failed to get metric statistics:
" + exception.getMessage(), exception);
                }
            });
    }


    /**
     * Retrieves and displays metric statistics for the specified parameters.
     *
     * @param nameSpace    the namespace for the metric
     * @param metVal       the name of the metric
     * @param metricOption the statistic to retrieve for the metric (e.g.,
"Maximum", "Average")
     * @param date         the date for which to retrieve the metric statistics,
in the format "yyyy-MM-dd'T'HH:mm:ss'Z'"
     * @param myDimension  the dimension(s) to filter the metric statistics by
     * @return a {@link CompletableFuture} that completes when the metric
statistics have been retrieved and displayed
     */
    public CompletableFuture<GetMetricStatisticsResponse>
getAndDisplayMetricStatisticsAsync(String nameSpace, String metVal,

            String metricOption, String date, Dimension myDimension) {

        Instant start = Instant.parse(date);
        Instant endDate = Instant.now();

        // Building the request for metric statistics.
```

```
        GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
            .endTime(endDate)
            .startTime(start)
            .dimensions(myDimension)
            .metricName(metVal)
            .namespace(nameSpace)
            .period(86400) // 1 day period
            .statistics(Statistic.fromValue(metricOption))
            .build();

        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {}",
datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    logger.info("Failed to get metric statistics: {} ",
exception.getMessage());
                }
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Error while getting metric
statistics: " + exception.getMessage(), exception);
            });
    }


    /**
     * Retrieves a list of metric names for the specified namespace.
     *
     * @param namespace the namespace for which to retrieve the metric names
     * @return a {@link CompletableFuture} that, when completed, contains an
{@link ArrayList} of
     * the metric names in the specified namespace
     * @throws RuntimeException if an error occurs while listing the metrics
     */
```

```
    public CompletableFuture<ArrayList<String>> listMetsAsync(String namespace) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        ListMetricsPublisher metricsPaginator =
getAsyncClient().listMetricsPaginator(request);
        Set<String> metSet = new HashSet<>();
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
            response.metrics().forEach(metric -> {
                String metricName = metric.metricName();
                metSet.add(metricName);
            });
        });

        return future
            .thenApply(ignored -> new ArrayList<>(metSet))
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list metrics: " +
exception.getMessage(), exception);
            });
    }

    /**
     * Lists the available namespaces for the current AWS account.
     *
     * @return a {@link CompletableFuture} that, when completed, contains an
{@link ArrayList} of the available namespace names.
     * @throws RuntimeException if an error occurs while listing the namespaces.
     */
    public CompletableFuture<ArrayList<String>> listNameSpacesAsync() {
        ArrayList<String> nameSpaceList = new ArrayList<>();
        ListMetricsRequest request = ListMetricsRequest.builder().build();

        ListMetricsPublisher metricsPaginator =
getAsyncClient().listMetricsPaginator(request);
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
            response.metrics().forEach(metric -> {
                String namespace = metric.namespace();
                if (!nameSpaceList.contains(namespace)) {
                    nameSpaceList.add(namespace);
                }
            });
        });
```

```
        return future
            .thenApply(ignored -> nameSpaceList)
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list namespaces: " +
exception.getMessage(), exception);
            });
    }
    /**
     * Retrieves the specific metric asynchronously.
     *
     * @param namespace the namespace of the metric to retrieve
     * @return a CompletableFuture that completes with the first dimension of the
first metric found in the specified namespace,
     * or throws a RuntimeException if an error occurs or no metrics or
dimensions are found
     */
    public CompletableFuture<Dimension> getSpecificMetAsync(String namespace) {
        ListMetricsRequest request = ListMetricsRequest.builder()
            .namespace(namespace)
            .build();

        return getAsyncClient().listMetrics(request).handle((response, exception)
-> {
            if (exception != null) {
                logger.info("Error occurred while listing metrics: {} ",
exception.getMessage());
                throw new RuntimeException("Failed to retrieve specific metric
dimension", exception);
            } else {
                List<Metric> myList = response.metrics();
                if (!myList.isEmpty()) {
                    Metric metric = myList.get(0);
                    if (!metric.dimensions().isEmpty()) {
                        return metric.dimensions().get(0); // Return the first
dimension
                    }
                }
                throw new RuntimeException("No metrics or dimensions found");
            }
        });
    }

    /**
```

```
    * Gets the current OTel enrichment status for the account. Enrichment is
what makes
    * CloudWatch attach AWS resource context to incoming OTLP metrics, so the
metrics
    * become correlatable with the rest of CloudWatch rather than opaque series.
    *
    * @return a {@link CompletableFuture} that completes with the status, such
as
    * {@code Running} or {@code NotStarted}
    */
   public CompletableFuture<String> getOTelEnrichmentStatusAsync() {
       return
getAsyncClient().getOTelEnrichment(GetOTelEnrichmentRequest.builder().build())
           .handle((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to get OTel enrichment
status: "
                           + exception.getMessage(), exception);
               }
               return response.statusAsString();
           });
   }

   /**
    * Turns on OTel enrichment for the account.
    *
    * @return a {@link CompletableFuture} that completes when enrichment has
started
    */
   public CompletableFuture<Void> startOTelEnrichmentAsync() {
       return
getAsyncClient().startOTelEnrichment(StartOTelEnrichmentRequest.builder().build())
           .handle((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to start OTel enrichment:
"
                           + exception.getMessage(), exception);
               }
               logger.info("Started OTel enrichment for this account.");
               return null;
           });
   }

   /**
```

```
    * Turns off OTel enrichment for the account. Existing PromQL alarms are not
deleted,
    * but vended metrics stop being enriched, so queries that select on the
added labels
    * stop matching.
    *
    * @return a {@link CompletableFuture} that completes when enrichment has
stopped
    */
   public CompletableFuture<Void> stopOTelEnrichmentAsync() {
       return
getAsyncClient().stopOTelEnrichment(StopOTelEnrichmentRequest.builder().build())
           .handle((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to stop OTel enrichment: "
                       + exception.getMessage(), exception);
               }
               logger.info("Stopped OTel enrichment for this account.");
               return null;
           });
   }

   /**
    * Creates an alarm that evaluates a PromQL query.
    *
    * <p>A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
    * match many series at once, and each matching series is tracked separately
as a
    * contributor. Instead of counting breaching periods, you specify durations:
a
    * contributor moves to ALARM after it breaches continuously for the pending
period,
    * and back to OK after it stops breaching for the recovery period. A PromQL
alarm
    * starts in the OK state rather than INSUFFICIENT_DATA.
    *
    * <p>{@link EvaluationCriteria} is a union and is mutually exclusive with
the classic
    * {@code metricName} and {@code metrics} parameters. When you use it you
must also set
    * {@code evaluationInterval}, and you must not set {@code period}, {@code
statistic},
```

```
    * {@code threshold}, {@code comparisonOperator}, or {@code
evaluationPeriods}.
    *
    * @param alarmName          the name of the alarm, unique within the Region
    * @param query              the PromQL query to evaluate. The comparison
belongs in
    *                           the query itself; there is no separate
threshold.
    * @param evaluationInterval how often, in seconds, to run the query. Valid
values are
    *                           10, 20, 30, and any multiple of 60, up to 3600.
    * @param pendingPeriod      how long, in seconds, a contributor must breach
    *                           continuously before it moves to ALARM
    * @param recoveryPeriod     how long, in seconds, a contributor must stop
breaching
    *                           before it moves back to OK
    * @return a {@link CompletableFuture} that completes when the alarm is
created
    */
   public CompletableFuture<Void> putPromQLMetricAlarmAsync(String alarmName,
String query,
           int evaluationInterval, int pendingPeriod, int recoveryPeriod) {
       AlarmPromQLCriteria promQLCriteria = AlarmPromQLCriteria.builder()
           .query(query)
           .pendingPeriod(pendingPeriod)
           .recoveryPeriod(recoveryPeriod)
           .build();

       PutMetricAlarmRequest request = PutMetricAlarmRequest.builder()
           .alarmName(alarmName)
           .alarmDescription("PromQL alarm created by the AWS SDK for Java 2.x
Basics scenario.")
           .evaluationCriteria(EvaluationCriteria.builder()
               .promQLCriteria(promQLCriteria)
               .build())
           .evaluationInterval(evaluationInterval)
           .build();

       return getAsyncClient().putMetricAlarm(request).handle((response,
exception) -> {
           if (exception != null) {
               throw new RuntimeException("Failed to create PromQL alarm: "
                   + exception.getMessage(), exception);
           }
```

```
            logger.info("Created PromQL alarm {} for query {}.", alarmName,
query);
            return null;
        });
    }

    /**
     * Gets the contributors for a PromQL alarm. Each contributor is one series
that the
     * alarm's query matched, identified by its label set. This is how you find
out which
     * hosts, services, or pods are breaching, rather than only that something
is.
     *
     * <p>The paging loop continues until the next token is empty. A page can
come back
     * empty while still carrying a next token, so stopping at the first empty
page would
     * silently drop later results.
     *
     * @param alarmName the name of the PromQL alarm
     * @return a {@link CompletableFuture} that completes with the list of
contributors,
     * which is empty when the query matched no series
     */
    public CompletableFuture<List<AlarmContributor>>
describeAlarmContributorsAsync(String alarmName) {
        List<AlarmContributor> contributors = new ArrayList<>();
        return collectContributorsPage(alarmName, null, contributors);
    }

    private CompletableFuture<List<AlarmContributor>>
collectContributorsPage(String alarmName,
            String nextToken, List<AlarmContributor> accumulated) {
        DescribeAlarmContributorsRequest request =
DescribeAlarmContributorsRequest.builder()
            .alarmName(alarmName)
            .nextToken(nextToken)
            .build();

        return getAsyncClient().describeAlarmContributors(request)
            .thenCompose(response -> {
                accumulated.addAll(response.alarmContributors());
                String token = response.nextToken();
```

```
                if (token == null || token.isEmpty()) {
                    return CompletableFuture.completedFuture(accumulated);
                }
                return collectContributorsPage(alarmName, token, accumulated);
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to describe alarm
 contributors: "
                    + exception.getMessage(), exception);
            });
    }

    /**
     * Creates or updates an alarm mute rule. While a mute rule is active the
 targeted
     * alarms keep evaluating and keep changing state, but their configured
 actions do not
     * fire. This is the supported way to suppress notifications during planned
     * maintenance, instead of disabling alarm actions and relying on someone to
 turn them
     * back on.
     *
     * @param name       the name of the mute rule
     * @param expression when the rule activates. For a recurring window, a five-
field
     *                   cron expression, {@code cron(Minutes Hours Day-of-month
 Month
     *                   Day-of-week)}, such as {@code cron(0 2 * * SUN)}. Note
 that this
     *                   is five fields, not the six that Amazon EventBridge
 uses. For a
     *                   one-time window, {@code at(yyyy-MM-ddThh:mm)}, such as
     *                   {@code at(2026-09-05T02:00)}, with no seconds.
     * @param duration   how long the window lasts once it activates, as an ISO
 8601
     *                   duration from {@code PT1M} to {@code P15D}. For example,
     *                   {@code PT2H} is two hours. Plain forms such as {@code
 2h} are
     *                   rejected.
     * @param timezone   a standard timezone identifier. Defaults to UTC when
 omitted.
     * @param alarmNames the names of up to 100 alarms to mute. If empty, the
 rule applies
     *                   to every alarm in the account.
```

```
    * @return a {@link CompletableFuture} that completes when the rule is
written
    */
   public CompletableFuture<Void> putAlarmMuteRuleAsync(String name, String
expression,
           String duration, String timezone, List<String> alarmNames) {
       Schedule schedule = Schedule.builder()
           .expression(expression)
           .duration(duration)
           .timezone(timezone)
           .build();

       PutAlarmMuteRuleRequest.Builder request =
PutAlarmMuteRuleRequest.builder()
           .name(name)
           .description("Mute rule created by the AWS SDK for Java 2.x Basics
scenario.")
           .rule(Rule.builder().schedule(schedule).build());

       if (alarmNames != null && !alarmNames.isEmpty()) {

request.muteTargets(MuteTargets.builder().alarmNames(alarmNames).build());
       }

       return
getAsyncClient().putAlarmMuteRule(request.build()).handle((response, exception)
-> {
           if (exception != null) {
               throw new RuntimeException("Failed to put alarm mute rule: "
                   + exception.getMessage(), exception);
           }
           logger.info("Put alarm mute rule {}.", name);
           return null;
       });
   }

   /**
    * Gets the full configuration of an alarm mute rule, including its schedule,
the
    * alarms it targets, and its current status.
    *
    * @param name the name of the mute rule
    * @return a {@link CompletableFuture} that completes with the mute rule
    */
```

```
   public CompletableFuture<GetAlarmMuteRuleResponse>
getAlarmMuteRuleAsync(String name) {
       return
getAsyncClient().getAlarmMuteRule(GetAlarmMuteRuleRequest.builder()
               .alarmMuteRuleName(name)
               .build())
           .handle((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to get alarm mute rule: "
                       + exception.getMessage(), exception);
               }
               return response;
           });
   }

   /**
    * Lists the alarm mute rules in the account, optionally filtered to the
rules that
    * target one alarm.
    *
    * <p>Note that {@link AlarmMuteRuleSummary} carries no name field, only an
ARN,
    * status, mute type, and last-updated timestamp. To find a rule by name,
match on the
    * ARN suffix.
    *
    * @param alarmName when non-null, only rules that target this alarm are
returned
    * @return a {@link CompletableFuture} that completes with the mute rule
summaries
    */
   public CompletableFuture<List<AlarmMuteRuleSummary>>
listAlarmMuteRulesAsync(String alarmName) {
       List<AlarmMuteRuleSummary> summaries = new ArrayList<>();
       return collectMuteRulesPage(alarmName, null, summaries);
   }

   private CompletableFuture<List<AlarmMuteRuleSummary>>
collectMuteRulesPage(String alarmName,
           String nextToken, List<AlarmMuteRuleSummary> accumulated) {
       ListAlarmMuteRulesRequest request = ListAlarmMuteRulesRequest.builder()
           .alarmName(alarmName)
           .nextToken(nextToken)
           .build();
```

```
        return getAsyncClient().listAlarmMuteRules(request)
            .thenCompose(response -> {
                accumulated.addAll(response.alarmMuteRuleSummaries());
                String token = response.nextToken();
                if (token == null || token.isEmpty()) {
                    return CompletableFuture.completedFuture(accumulated);
                }
                return collectMuteRulesPage(alarmName, token, accumulated);
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list alarm mute rules: "
                    + exception.getMessage(), exception);
            });
    }

    /**
     * Deletes an alarm mute rule. The alarms it targeted resume firing their
 actions.
     *
     * @param name the name of the mute rule
     * @return a {@link CompletableFuture} that completes when the rule is
 deleted
     */
    public CompletableFuture<Void> deleteAlarmMuteRuleAsync(String name) {
        return
 getAsyncClient().deleteAlarmMuteRule(DeleteAlarmMuteRuleRequest.builder()
                .alarmMuteRuleName(name)
                .build())
            .handle((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete alarm mute rule:
 "
                        + exception.getMessage(), exception);
                }
                logger.info("Deleted alarm mute rule {}.", name);
                return null;
            });
    }

    public static String readFileAsString(String file) throws IOException {
        return new String(Files.readAllBytes(Paths.get(file)));
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetDashboard
  - GetMetricStatistics
  - GetOTelEnrichment
  - ListAlarmMuteRules
  - ListDashboards
  - ListMetrics
  - PutAlarmMuteRule
  - PutDashboard
  - PutMetricAlarm
  - StartOTelEnrichment
  - StopOTelEnrichment

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating the CloudWatch OpenTelemetry experience.

```
import {
  Scenario,
  ScenarioAction,
```

```
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  CloudWatchClient,
  DeleteAlarmMuteRuleCommand,
  DeleteAlarmsCommand,
  DeleteDashboardsCommand,
  DescribeAlarmContributorsCommand,
  GetAlarmMuteRuleCommand,
  GetDashboardCommand,
  GetMetricStatisticsCommand,
  GetOTelEnrichmentCommand,
  ListAlarmMuteRulesCommand,
  ListMetricsCommand,
  PutAlarmMuteRuleCommand,
  PutDashboardCommand,
  PutMetricAlarmCommand,
  StartOTelEnrichmentCommand,
  StopOTelEnrichmentCommand,
} from "@aws-sdk/client-cloudwatch";
import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";

const DEFAULT_QUERY = "avg by (host) (system_cpu_utilization) > 80";

// Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to 3600
 seconds.
const EVALUATION_INTERVAL = 60;
const PENDING_PERIOD = 300;
const RECOVERY_PERIOD = 120;

/**
 * @typedef {{
 *   client: import('@aws-sdk/client-cloudwatch').CloudWatchClient,
 *   alarmName: string,
 *   dashboardName: string,
 *   muteRuleName: string,
 *   namespaces: [string, number][],
 *   metric: import('@aws-sdk/client-cloudwatch').Metric | undefined,
 *   query: string,
 *   startedEnrichment: boolean,
 *   dashboardCreated: boolean,
 *   deleteResources: boolean,
```

```
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `Welcome to the Amazon CloudWatch Basics scenario.

CloudWatch now ingests OpenTelemetry metrics natively. This scenario walks
 through that experience: it turns on OTel enrichment so CloudWatch can correlate
 incoming OTLP metrics with the resources that produced them, alarms on those
 metrics with a PromQL query, and shows you which individual series drove the
 alarm.

A PromQL alarm works differently from a classic metric alarm. Rather than
 watching one metric and counting breaching periods, it evaluates a query that
 can match many series at once, and tracks each one separately as a contributor.

Note that sending OTLP metrics to CloudWatch is not an AWS SDK operation. Metrics
 arrive over the OTLP protocol through the CloudWatch agent, an OpenTelemetry
 Collector, or an ADOT SDK. Everything this scenario does is configuration and
 querying around that ingestion path.

Let's get started...`,
  { header: true },
);

// Step 1: List metrics and namespaces. This orients the reader before any
 configuration
// happens.
const displayListMetrics = new ScenarioOutput(
  "displayListMetrics",
  "1. List metrics and namespaces\n\nBefore configuring anything, let's see what
 CloudWatch is already collecting in this account by calling ListMetrics.",
);

const sdkListMetrics = new ScenarioAction(
```

```
  "sdkListMetrics",
  async (/** @type {State} */ state) => {
    const counts = new Map();
    let metricCount = 0;
    let nextToken;

    do {
      const response = await state.client.send(
        new ListMetricsCommand({ NextToken: nextToken }),
      );
      for (const metric of response.Metrics ?? []) {
        counts.set(metric.Namespace, (counts.get(metric.Namespace) ?? 0) + 1);
        metricCount += 1;
        // Keep the first metric we see so later steps have something to chart.
        if (!state.metric) {
          state.metric = metric;
        }
      }
      nextToken = response.NextToken;
      // This account may have a very large number of metrics, so stop once we
 have
      // enough to give the reader a sense of what is there.
    } while (nextToken && metricCount < 500);

    state.namespaces = [...counts.entries()].sort((a, b) => b[1] - a[1]);

    console.log(
      `\tFound ${metricCount} metrics across ${state.namespaces.length}
 namespaces:`,
    );
    for (const [namespace, count] of state.namespaces.slice(0, 10)) {
      console.log(`\t  ${namespace} (${count} metrics)`);
    }
    if (state.namespaces.length === 0) {
      console.log(
        "\tNo metrics found in this account. The statistics and dashboard steps
 later on need an existing metric, so they will be skipped.",
      );
    }
  },
);

// Step 2: Start OTel enrichment. Enrichment is what makes CloudWatch attach AWS
 resource
```

```
// context to incoming OTLP metrics.
const displayStartEnrichment = new ScenarioOutput(
  "displayStartEnrichment",
  "2. Start OpenTelemetry enrichment\n\nEnrichment is what lets CloudWatch attach
 AWS resource context to the OTLP metrics you send it. Without it, your metrics
 arrive as opaque series with no connection to the resources that emitted them.\n
\nWe check the current state first, and only start enrichment if it isn't already
 on.",
);

const sdkStartEnrichment = new ScenarioAction(
  "sdkStartEnrichment",
  async (/** @type {State} */ state) => {
    const { Status } = await state.client.send(
      new GetOTelEnrichmentCommand({}),
    );
    console.log(`\tEnrichment status: ${Status}`);

    if (Status === "Running") {
      console.log(
        "\n\tEnrichment was already running, so we will leave it alone. The
 cleanup step will not stop it, because other workloads in this account may
 depend on it.",
      );
      return;
    }

    await state.client.send(new StartOTelEnrichmentCommand({}));
    // Record that *this run* started enrichment, so cleanup only stops what it
 turned on.
    state.startedEnrichment = true;

    const after = await state.client.send(new GetOTelEnrichmentCommand({}));
    console.log(`\tEnrichment status: ${after.Status}`);
    console.log(
      "\n\tNote: this run started enrichment, so the cleanup step will stop it
 again.",
    );
  },
);

// Step 3: Explain OTLP ingestion. This step makes no service call; naming the
 gap
// explicitly is the point.
```

```
const displayOtlpIngestion = new ScenarioOutput(
  "displayOtlpIngestion",
  `3. Send OTLP metrics to CloudWatch

This step is not an AWS SDK operation, and that's worth being explicit about.
 Metrics reach CloudWatch over the OTLP protocol, through the CloudWatch agent,
 an OpenTelemetry Collector, or an ADOT SDK. There is no PutOTelMetrics API to
 call.

Point your collector at the CloudWatch metrics endpoint, which follows the
 pattern
\thttps://monitoring.<region>.amazonaws.com/v1/metrics

The endpoint is HTTP/1.1 only and does not support gRPC, so use an otlphttp
 exporter rather than otlp. The metrics endpoint signs as "monitoring".`,
);

// Step 4: Create a PromQL alarm.
const displayCreateAlarm = new ScenarioOutput(
  "displayCreateAlarm",
  "4. Create a PromQL alarm\n\nNow we alarm on those metrics. The comparison goes
 inside the query itself: a PromQL alarm has no separate threshold, comparison
 operator, statistic, or period.",
);

const inputQuery = new ScenarioInput("query", "Enter a PromQL query:", {
  type: "input",
  default: DEFAULT_QUERY,
});

const sdkCreateAlarm = new ScenarioAction(
  "sdkCreateAlarm",
  async (/** @type {State} */ state) => {
    const query = state.query?.trim() || DEFAULT_QUERY;
    state.query = query;

    // EvaluationCriteria is a union and is mutually exclusive with the classic
 MetricName
    // and Metrics parameters. When you use it you must also set
 EvaluationInterval, and
    // you must not set Period, Statistic, Threshold, ComparisonOperator,
    // EvaluationPeriods, DatapointsToAlarm, or TreatMissingData.
    await state.client.send(
      new PutMetricAlarmCommand({
```

```
        AlarmName: state.alarmName,
        AlarmDescription:
          "A PromQL alarm created by the AWS SDK for JavaScript Basics
 scenario.",
        EvaluationCriteria: {
          PromQLCriteria: {
            Query: query,
            PendingPeriod: PENDING_PERIOD,
            RecoveryPeriod: RECOVERY_PERIOD,
          },
        },
        EvaluationInterval: EVALUATION_INTERVAL,
        ActionsEnabled: false,
      }),
    );

    console.log(`\tCreated alarm ${state.alarmName}:`);
    console.log(`\t  query:              ${query}`);
    console.log(`\t  evaluationInterval: ${EVALUATION_INTERVAL} seconds`);
    console.log(`\t  pendingPeriod:      ${PENDING_PERIOD} seconds`);
    console.log(`\t  recoveryPeriod:     ${RECOVERY_PERIOD} seconds`);
    console.log(
      "\n\tA PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA,
 which is another way it differs from a classic alarm.",
    );
  },
);

// Step 5: Inspect the alarm's contributors. This is the step with no classic-
alarm
// equivalent.
const displayContributors = new ScenarioOutput(
  "displayContributors",
  "5. Inspect the alarm's contributors\n\nEach contributor is one series the
 query matched, identified by its label set. This is how you find out which
 host is unhealthy rather than only that something is. Classic alarms have no
 equivalent.",
);

const sdkContributors = new ScenarioAction(
  "sdkContributors",
  async (/** @type {State} */ state) => {
    const contributors = [];
    let nextToken;
```

```
    do {
      const response = await state.client.send(
        new DescribeAlarmContributorsCommand({
          AlarmName: state.alarmName,
          NextToken: nextToken,
        }),
      );
      contributors.push(...(response.AlarmContributors ?? []));
      nextToken = response.NextToken;
      // A page can come back empty while still carrying a token, so keep going
 until the
      // token itself is gone rather than stopping at the first empty page.
    } while (nextToken);

    if (contributors.length === 0) {
      console.log(
        "\tNo contributors yet. The query matched no series, which usually means
 no OTel metrics with these labels have arrived. Once your collector is sending
 data, each matching series appears here with its labels and the reason it
 breached.",
      );
      return;
    }

    console.log(`\tFound ${contributors.length} contributors:`);
    for (const contributor of contributors) {
      const labels = Object.entries(contributor.ContributorAttributes ?? {})
        .sort(([a], [b]) => a.localeCompare(b))
        .map(([key, value]) => `${key}=${value}`)
        .join(", ");
      console.log(`\t  ${contributor.ContributorId}: ${labels}`);
      console.log(`\t    reason: ${contributor.StateReason}`);
    }
  },
);

// Step 6: Get statistics and chart the metric on a dashboard.
const displayDashboard = new ScenarioOutput(
  "displayDashboard",
  "6. Get statistics and chart the metric on a dashboard\n\nStatistics and
 dashboards are how you see what the alarm is evaluating.",
);
```

```
const sdkDashboard = new ScenarioAction(
  "sdkDashboard",
  async (/** @type {State} */ state) => {
    if (!state.metric) {
      console.log(
        "\tSkipping statistics and dashboard because no metrics exist yet.",
      );
      return;
    }

    const metric = state.metric;
    const stats = await state.client.send(
      new GetMetricStatisticsCommand({
        Namespace: metric.Namespace,
        MetricName: metric.MetricName,
        Dimensions: metric.Dimensions,
        StartTime: new Date(Date.now() - 24 * 60 * 60 * 1000),
        EndTime: new Date(),
        Period: 3600,
        Statistics: ["Average", "Maximum"],
      }),
    );

    const datapoints = stats.Datapoints ?? [];
    console.log(
      `\tStatistics for ${metric.Namespace} ${metric.MetricName} over the last
 day:`,
    );
    console.log(`\t  Datapoints: ${datapoints.length}`);
    for (const datapoint of datapoints.slice(0, 3)) {
      console.log(
        `\t  ${datapoint.Timestamp?.toISOString()} average ${datapoint.Average},
 maximum ${datapoint.Maximum}`,
      );
    }

    const region = await state.client.config.region();
    const response = await state.client.send(
      new PutDashboardCommand({
        DashboardName: state.dashboardName,
        DashboardBody: buildDashboardBody(metric, region),
      }),
    );
    state.dashboardCreated = true;
```

```
    for (const message of response.DashboardValidationMessages ?? []) {
      console.log(`\tDashboard validation message: ${message.Message}`);
    }
    console.log(`\tCreated dashboard ${state.dashboardName}.`);

    const stored = await state.client.send(
      new GetDashboardCommand({ DashboardName: state.dashboardName }),
    );
    console.log(
      `\tRead the dashboard back, ${stored.DashboardBody?.length} characters of
 widget JSON.`,
    );
  },
);

/**
 * Build a single-widget dashboard body that charts the given metric.
 * @param {import('@aws-sdk/client-cloudwatch').Metric} metric
 * @param {string} region The region the metric is in. A metric widget must name
 *   its region, because a dashboard can chart metrics from several.
 * @returns {string} The dashboard body, as JSON.
 */
const buildDashboardBody = (metric, region) => {
  const metricSpec = [metric.Namespace, metric.MetricName];
  for (const dimension of metric.Dimensions ?? []) {
    metricSpec.push(dimension.Name, dimension.Value);
  }

  return JSON.stringify({
    widgets: [
      {
        type: "text",
        x: 0,
        y: 0,
        width: 24,
        height: 2,
        properties: {
          markdown:
            "This dashboard was created programmatically by an AWS SDK code
 example.",
        },
      },
      {
```

```
        type: "metric",
        x: 0,
        y: 2,
        width: 12,
        height: 6,
        properties: {
          metrics: [metricSpec],
          view: "timeSeries",
          stat: "Average",
          period: 300,
          region,
          title: metric.MetricName,
        },
      },
    ],
  });
};

// Step 7: Mute the alarm for a maintenance window.
const displayMuteRule = new ScenarioOutput(
  "displayMuteRule",
  "7. Mute the alarm for a maintenance window\n\nWhile a mute rule is active the
 targeted alarms keep evaluating and keep changing state, but their actions do
 not fire. This is the supported way to suppress notifications during planned
 maintenance, instead of disabling alarm actions and hoping someone remembers to
 turn them back on.",
);

const sdkMuteRule = new ScenarioAction(
  "sdkMuteRule",
  async (/** @type {State} */ state) => {
    // The expression is a five-field cron expression,
    // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is five
 fields,
    // not the six that Amazon EventBridge uses. For a one-time window, use
    // at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
 duration from
    // PT1M to P15D, so PT2H rather than 2h.
    const expression = "cron(0 2 * * SUN)";
    const duration = "PT2H";
    const timezone = "America/Los_Angeles";

    await state.client.send(
      new PutAlarmMuteRuleCommand({
```

```
        Name: state.muteRuleName,
        Description:
          "A mute rule created by the AWS SDK for JavaScript Basics scenario.",
        Rule: {
          Schedule: {
            Expression: expression,
            Duration: duration,
            Timezone: timezone,
          },
        },
        // Target up to 100 alarms. If MuteTargets is omitted, the rule applies
to every
        // alarm in the account.
        MuteTargets: { AlarmNames: [state.alarmName] },
      }),
    );

    console.log(`\tCreated mute rule ${state.muteRuleName}:`);
    console.log(`\t  schedule: ${expression} for ${duration}`);
    console.log(`\t  timezone: ${timezone}`);
    console.log(`\t  targets:  ${state.alarmName}`);
    console.log(
      "\n\tNote the two formats here. The expression is a five-field cron
expression, five rather than the six Amazon EventBridge uses. The duration is an
ISO 8601 duration, so 'PT2H' and not '2h'.",
    );
    console.log(
      "\n\tAlso note that MuteTargets is set explicitly. If you leave it out, the
rule applies to every alarm in the account.",
    );

    const rule = await state.client.send(
      new GetAlarmMuteRuleCommand({ AlarmMuteRuleName: state.muteRuleName }),
    );
    console.log(
      `\tRead the rule back: status ${rule.Status}, mute type ${rule.MuteType}.`,
    );

    const summaries = [];
    let nextToken;
    do {
      const response = await state.client.send(
        new ListAlarmMuteRulesCommand({
          AlarmName: state.alarmName,
```

```
          NextToken: nextToken,
        }),
      );
      summaries.push(...(response.AlarmMuteRuleSummaries ?? []));
      nextToken = response.NextToken;
    } while (nextToken);

    console.log(`\tFound ${summaries.length} mute rules targeting this alarm.`);
    // Mute rule summaries carry no name field, only an ARN, so match on the ARN
 suffix.
    const match = summaries.find(
      (summary) =>
        summary.AlarmMuteRuleArn?.endsWith(`/${state.muteRuleName}`) ||
        summary.AlarmMuteRuleArn?.endsWith(`:${state.muteRuleName}`),
    );
    if (match) {
      console.log(
        `\t  matched by ARN: ${match.AlarmMuteRuleArn} (${match.Status})`,
      );
    }
  },
);

// Step 8: Clean up.
const askToDeleteResources = new ScenarioInput(
  "deleteResources",
  "8. Clean up\n\nDelete the resources this scenario created?",
  { type: "confirm" },
);

const displaySkipCleanUp = new ScenarioOutput(
  "displaySkipCleanUp",
  "\tSkipping cleanup. Note that the alarm, dashboard, and mute rule are still in
 your account, and enrichment may still be running.",
  { skipWhen: (/** @type {State} */ state) => state.deleteResources },
);

const sdkCleanUp = new ScenarioAction(
  "sdkCleanUp",
  async (/** @type {State} */ state) => {
    // Each deletion is attempted independently so that one failure does not
 leave the
    // remaining resources behind.
    try {
```

```
      await state.client.send(
        new DeleteAlarmMuteRuleCommand({
          AlarmMuteRuleName: state.muteRuleName,
        }),
      );
      console.log(`\tDeleted mute rule ${state.muteRuleName}.`);
    } catch (caught) {
      console.log(`\tCould not delete the mute rule: ${caught.message}`);
    }

    try {
      await state.client.send(
        new DeleteAlarmsCommand({ AlarmNames: [state.alarmName] }),
      );
      console.log(`\tDeleted alarm ${state.alarmName}.`);
    } catch (caught) {
      console.log(`\tCould not delete the alarm: ${caught.message}`);
    }

    if (state.dashboardCreated) {
      try {
        await state.client.send(
          new DeleteDashboardsCommand({
            DashboardNames: [state.dashboardName],
          }),
        );
        console.log(`\tDeleted dashboard ${state.dashboardName}.`);
      } catch (caught) {
        console.log(`\tCould not delete the dashboard: ${caught.message}`);
      }
    }

    if (!state.startedEnrichment) {
      console.log(
        "\tLeft OTel enrichment running, because it was already on before this
  run.",
      );
      return;
    }

    try {
      await state.client.send(new StopOTelEnrichmentCommand({}));
      console.log("\tStopped OTel enrichment, because this run started it.");
    } catch (caught) {
```

```
      console.log(`\tCould not stop OTel enrichment: ${caught.message}`);
    }
  },
  { skipWhen: (/** @type {State} */ state) => !state.deleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the Amazon CloudWatch Basics scenario.",
);

// Suffix the resource names so repeated runs do not collide.
const suffix = Math.floor(Math.random() * 9000) + 1000;

const myScenario = new Scenario(
  "CloudWatch Basics",
  [
    greet,
    pressEnter,
    displayListMetrics,
    sdkListMetrics,
    pressEnter,
    displayStartEnrichment,
    sdkStartEnrichment,
    pressEnter,
    displayOtlpIngestion,
    pressEnter,
    displayCreateAlarm,
    inputQuery,
    sdkCreateAlarm,
    pressEnter,
    displayContributors,
    sdkContributors,
    pressEnter,
    displayDashboard,
    sdkDashboard,
    pressEnter,
    displayMuteRule,
    sdkMuteRule,
    pressEnter,
    askToDeleteResources,
    displaySkipCleanUp,
    sdkCleanUp,
    goodbye,
```

```
  ],
  {
    client: new CloudWatchClient({}),
    alarmName: `doc-example-promql-alarm-${suffix}`,
    dashboardName: `doc-example-dashboard-${suffix}`,
    muteRuleName: `doc-example-mute-rule-${suffix}`,
    namespaces: [],
    metric: undefined,
    startedEnrichment: false,
    dashboardCreated: false,
  },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetDashboard
  - GetMetricStatistics

- GetOTelEnrichment
- ListAlarmMuteRules
- ListDashboards
- ListMetrics
- PutAlarmMuteRule
- PutDashboard
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating the CloudWatch OpenTelemetry experience.

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:
 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 This scenario demonstrates the Amazon CloudWatch OpenTelemetry (OTel)
 experience.
 CloudWatch ingests OpenTelemetry metrics natively, and this example walks
 through what
 you do with them: turning on enrichment so CloudWatch can correlate incoming
 OTLP
 metrics with the resources that produced them, alarming on those metrics with a
 PromQL
```

```
 query, and finding out which individual series drove the alarm.

 A PromQL alarm works differently from a classic metric alarm. Rather than
 watching one
 metric and counting breaching periods, it evaluates a query that can match many
 series
 at once, and tracks each matching series separately as a contributor.

 Note that sending OTLP metrics to CloudWatch is not an AWS SDK operation.
 Metrics
 arrive over the OTLP protocol through the CloudWatch agent, an OpenTelemetry
 Collector, or an ADOT SDK. Everything this scenario does is configuration and
 querying
 around that ingestion path.

 This Kotlin code example performs the following tasks:

 1. List metrics and namespaces from Amazon CloudWatch.
 2. Start OpenTelemetry enrichment for the account.
 3. Explain how OTLP metrics reach CloudWatch.
 4. Create an alarm that evaluates a PromQL query.
 5. Inspect the contributors to the PromQL alarm.
 6. Get metric statistics and chart the metric on a dashboard.
 7. Mute the alarm for a maintenance window.
 8. Clean up the Amazon CloudWatch resources.
 */

val DASHES: String = "-".repeat(80)

private const val REGION = "us-east-1"

private const val DEFAULT_QUERY = "avg by (host) (system_cpu_utilization) > 80"

// Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to 3600
 seconds.
private const val EVALUATION_INTERVAL = 60
private const val PENDING_PERIOD = 300
private const val RECOVERY_PERIOD = 120

val scenarioScanner = Scanner(System.`in`)

suspend fun main() {
    // Suffix the resource names so repeated runs do not collide.
    val suffix = (Random().nextInt(9000) + 1000).toString()
```

```
    val alarmName = "doc-example-promql-alarm-$suffix"
    val dashboardName = "doc-example-dashboard-$suffix"
    val muteRuleName = "doc-example-mute-rule-$suffix"

    println(DASHES)
    println("Welcome to the Amazon CloudWatch Basics scenario.")
    println(
        """
        CloudWatch now ingests OpenTelemetry metrics natively. This scenario
walks through
        that experience: it turns on OTel enrichment so CloudWatch can correlate
incoming
        OTLP metrics with the resources that produced them, alarms on those
metrics with a
        PromQL query, and shows you which individual series drove the alarm.

        A PromQL alarm works differently from a classic metric alarm. Rather than
watching
        one metric and counting breaching periods, it evaluates a query that can
match many
        series at once, and tracks each one separately as a contributor.

        Let's get started...
        """.trimIndent(),
    )
    waitForInputToContinue()

    // Tracks whether this run turned enrichment on, so that cleanup only turns
off
    // enrichment that this run started.
    var startedEnrichment = false
    var dashboardCreated = false

    println(DASHES)
    println(
        """
        1. List metrics and namespaces

        Before configuring anything, let's see what CloudWatch is already
collecting in
        this account by calling ListMetrics.
        """.trimIndent(),
    )
    waitForInputToContinue()
```

```
    val namespaces = listNameSpaces()
    println("Found ${namespaces.size} namespaces in this account:")
    namespaces.take(10).forEach { println("  $it") }
    if (namespaces.isEmpty()) {
        println(
            """
            No metrics found in this account. The statistics and dashboard steps
later on
            need an existing metric, so they will be skipped.
            """.trimIndent(),
        )
    }
    waitForInputToContinue()

    println(DASHES)
    println(
        """
        2. Start OpenTelemetry enrichment

        Enrichment is what lets CloudWatch attach AWS resource context to the
OTLP metrics
        you send it. Without it, your metrics arrive as opaque series with no
connection to
        the resources that emitted them.

        We check the current state first, and only start enrichment if it isn't
already on.
        """.trimIndent(),
    )
    waitForInputToContinue()

    val status = getOTelEnrichmentStatus()
    if (status !is OTelEnrichmentStatus.Running) {
        // Record the attempt before making it. We already know enrichment was
not running, so
        // stopping it during cleanup is always safe, and a start that succeeds
but fails to
        // report back would otherwise leave it running.
        startedEnrichment = true
        startOTelEnrichment()

        val newStatus = getOTelEnrichmentStatus()
        println(
```

```
            "Note: this run started enrichment (status is now
${newStatus?.value}), so the " +
                "cleanup step will stop it again.",
        )
    } else {
        println(
            """
            Enrichment was already running, so we will leave it alone. The
cleanup step
            will not stop it, because other workloads in this account may depend
on it.
            """.trimIndent(),
        )
    }
    waitForInputToContinue()

    println(DASHES)
    println(
        """
        3. Send OTLP metrics to CloudWatch

        This step is not an AWS SDK operation, and that's worth being explicit
about.
        Metrics reach CloudWatch over the OTLP protocol, through the CloudWatch
agent, an
        OpenTelemetry Collector, or an ADOT SDK. There is no PutOTelMetrics API
to call.

        Point your collector at the CloudWatch metrics endpoint, which follows
the pattern
        https://monitoring.<region>.amazonaws.com/v1/metrics

        The endpoint is HTTP/1.1 only and does not support gRPC, so use an
otlphttp
        exporter rather than otlp. The metrics endpoint signs as "monitoring".
        """.trimIndent(),
    )
    waitForInputToContinue()

    println(DASHES)
    println(
        """
        4. Create a PromQL alarm
```

```
        Now we alarm on those metrics. The comparison goes inside the query
itself: a
        PromQL alarm has no separate threshold, comparison operator, statistic,
or period.
        """.trimIndent(),
    )
    println("Enter a PromQL query, or press <ENTER> for the default")
    println("[$DEFAULT_QUERY]:")
    val queryInput = scenarioScanner.nextLine()
    val query = if (queryInput.isBlank()) DEFAULT_QUERY else queryInput.trim()

    putPromQlMetricAlarm(alarmName, query, EVALUATION_INTERVAL, PENDING_PERIOD,
RECOVERY_PERIOD)
    println("Created alarm $alarmName:")
    println("  query:              $query")
    println("  evaluationInterval: $EVALUATION_INTERVAL seconds")
    println("  pendingPeriod:      $PENDING_PERIOD seconds")
    println("  recoveryPeriod:     $RECOVERY_PERIOD seconds")
    println(
        """
        A PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA,
which is
        another way it differs from a classic alarm.
        """.trimIndent(),
    )
    waitForInputToContinue()

    println(DASHES)
    println(
        """
        5. Inspect the alarm's contributors

        Each contributor is one series the query matched, identified by its label
set. This
        is how you find out which host is unhealthy rather than only that
something is.
        Classic alarms have no equivalent.
        """.trimIndent(),
    )
    waitForInputToContinue()

    describeAlarmContributors(alarmName)
    waitForInputToContinue()
```

```
    println(DASHES)
    println(
        """
        6. Get statistics and chart the metric on a dashboard

        Statistics and dashboards are how you see what the alarm is evaluating.
        """.trimIndent(),
    )
    waitForInputToContinue()

    if (namespaces.isNotEmpty()) {
        val namespace = namespaces[0]
        val metrics = listMets(namespace)
        if (metrics != null && metrics.isNotEmpty()) {
            val metricName = metrics[0]
            val startDate = Instant.now().minus(24, ChronoUnit.HOURS).toString()
            var dimension: Dimension? = null
            try {
                dimension = getSpecificMet(namespace)
                if (dimension != null) {
                    getAndDisplayMetricStatistics(namespace, metricName,
"Average", startDate, dimension)
                }
            } catch (e: Exception) {
                println("Could not get statistics for $namespace/$metricName:
${e.message}")
            }

            // Chart the metric this run just discovered. Reading the widgets
from a file
            // would chart metrics that may not exist in this account.
            try {
                createDashboard(dashboardName, buildDashboardBody(namespace,
metricName, dimension, REGION))
                dashboardCreated = true
                listDashboards()
            } catch (e: Exception) {
                println("Could not create the dashboard: ${e.message}")
            }
        } else {
            println("No metrics found in namespace $namespace, skipping
statistics and the dashboard.")
        }
    } else {
```

```
        println("Skipping statistics and dashboard because no metrics exist
yet.")
    }
    waitForInputToContinue()

    println(DASHES)
    println(
        """
        7. Mute the alarm for a maintenance window

        While a mute rule is active the targeted alarms keep evaluating and keep
changing
        state, but their actions do not fire. This is the supported way to
suppress
        notifications during planned maintenance, instead of disabling alarm
actions and
        hoping someone remembers to turn them back on.
        """.trimIndent(),
    )
    waitForInputToContinue()

    // The expression is a five-field cron expression,
    // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is five
fields,
    // not the six that Amazon EventBridge uses. For a one-time window, use
    // at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
duration from
    // PT1M to P15D, so PT2H rather than 2h.
    val expression = "cron(0 2 * * SUN)"
    val duration = "PT2H"
    val timezone = "America/Los_Angeles"

    putAlarmMuteRule(muteRuleName, expression, duration, listOf(alarmName),
timezone)
    println("Created mute rule $muteRuleName:")
    println("  schedule: $expression for $duration")
    println("  timezone: $timezone")
    println("  targets:  $alarmName")
    println(
        """
        Note the two formats here. The expression is a five-field cron
expression, five
        rather than the six Amazon EventBridge uses. The duration is an ISO 8601
duration,
```

```
        so 'PT2H' and not '2h'.

        Also note that muteTargets is set explicitly. If you leave it out, the
rule applies
        to every alarm in the account.
        """.trimIndent(),
    )

    val muteRule = getAlarmMuteRule(muteRuleName)
    println("Read the rule back: status ${muteRule.status?.value}, mute type
${muteRule.muteType}.")

    val summaries = listAlarmMuteRules(alarmName)
    println("Found ${summaries.size} mute rules targeting this alarm.")
    // Mute rule summaries carry no name field, only an ARN, so match on the ARN
suffix.
    summaries
        .firstOrNull { summary ->
            summary.alarmMuteRuleArn?.endsWith("/$muteRuleName") == true ||
                summary.alarmMuteRuleArn?.endsWith(":$muteRuleName") == true
        }?.let { summary ->
            println("  matched by ARN: ${summary.alarmMuteRuleArn}
(${summary.status?.value})")
        }
    waitForInputToContinue()

    println(DASHES)
    println("8. Clean up")
    println("Delete the resources this scenario created? (y/n)")
    val cleanUp = scenarioScanner.nextLine()
    if (!cleanUp.trim().equals("y", ignoreCase = true)) {
        println(
            """
            Skipping cleanup. Note that the alarm, dashboard, and mute rule are
still in
            your account, and enrichment may still be running.
            """.trimIndent(),
        )
        println(DASHES)
        println("This concludes the Amazon CloudWatch Basics scenario.")
        return
    }
```

```
    // Each deletion is attempted independently so that one failure does not
 leave the
    // remaining resources behind.
    try {
        deleteAlarmMuteRule(muteRuleName)
    } catch (e: Exception) {
        println("Could not delete the mute rule: ${e.message}")
    }

    try {
        deleteAlarm(alarmName)
    } catch (e: Exception) {
        println("Could not delete the alarm: ${e.message}")
    }

    if (dashboardCreated) {
        try {
            deleteDashboard(dashboardName)
        } catch (e: Exception) {
            println("Could not delete the dashboard: ${e.message}")
        }
    }

    if (startedEnrichment) {
        try {
            stopOTelEnrichment()
            println("Stopped OTel enrichment, because this run started it.")
        } catch (e: Exception) {
            println("Could not stop OTel enrichment: ${e.message}")
        }
    } else {
        println("Left OTel enrichment running, because it was already on before
 this run.")
    }

    println(DASHES)
    println("This concludes the Amazon CloudWatch Basics scenario.")
    println(DASHES)
}

private fun waitForInputToContinue() {
    while (true) {
        println("")
        println("Press <ENTER> to continue:")
```

```
        val input = scenarioScanner.nextLine()
        if (input.trim().isEmpty()) {
            println("Continuing with the program...")
            println("")
            break
        }
        println("Invalid input. Please try again.")
    }
}

suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}

suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}

suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}
```

```
suspend fun createDashboard(
    dashboardNameVal: String,
    dashboardBodyVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = dashboardBodyVal
        }

    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
                    println("Message is: ${message.message}")
                }
            }
        }
    }
}

/**
 * Builds a single-widget dashboard body that charts the given metric.
 *
 * A metric widget must name its Region, because a dashboard can chart metrics
 from several.
 */
fun buildDashboardBody(
    metricNamespace: String,
    metricName: String,
    dimension: Dimension?,
    region: String,
): String {
    val dimensionParts =
        if (dimension == null) "" else ", \"${dimension.name}\",
 \"${dimension.value}\""

    return """
```

```
        {
            "widgets": [
                {
                    "type": "text",
                    "x": 0, "y": 0, "width": 24, "height": 2,
                    "properties": {
                        "markdown": "This dashboard was created programmatically
 by an AWS SDK code example."
                    }
                },
                {
                    "type": "metric",
                    "x": 0, "y": 2, "width": 12, "height": 6,
                    "properties": {
                        "metrics": [[ "$metricNamespace",
 "$metricName"$dimensionParts ]],
                        "view": "timeSeries",
                        "stat": "Average",
                        "period": 300,
                        "region": "$region",
                        "title": "$metricName"
                    }
                }
            ]
        }
    """.trimIndent()
}

suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
```

```
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}

suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}

suspend fun getSpecificMet(namespaceVal: String?): Dimension? {
    val request =
```

```
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val response = cwClient.listMetrics(request)
        val myList = response.metrics
        if (myList != null) {
            return myList[0].dimensions?.get(0)
        }
    }
    return null
}

suspend fun listNameSpaces(): ArrayList<String> {
    val nameSpaceList = ArrayList<String>()
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val response = cwClient.listMetrics(ListMetricsRequest {})
        response.metrics?.forEach { metrics ->
            val data = metrics.namespace
            if (!nameSpaceList.contains(data)) {
                nameSpaceList.add(data!!)
            }
        }
    }
    return nameSpaceList
}
```

The OpenTelemetry functions that the scenario calls.

```
suspend fun getOTelEnrichmentStatus(): OTelEnrichmentStatus? {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getOTelEnrichment(GetOTelEnrichmentRequest {})
        val status = response.status
        when (status) {
            is OTelEnrichmentStatus.Running ->
                println("OTel enrichment is running. Vended metrics are queryable
 with PromQL")
            is OTelEnrichmentStatus.Stopped ->
                println("OTel enrichment is stopped. Start it to enrich vended
 metrics")
            else -> println("OTel enrichment status is ${status?.value}")
        }
```

```
        return status
    }
}

suspend fun startOTelEnrichment() {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.startOTelEnrichment(StartOTelEnrichmentRequest {})
        println("Successfully started OTel enrichment for this account")
    }
}

suspend fun putPromQlMetricAlarm(
    alarmNameVal: String,
    queryVal: String,
    evaluationIntervalVal: Int = 60,
    pendingPeriodVal: Int = 300,
    recoveryPeriodVal: Int = 120,
) {
    // The comparison belongs in the query itself. A PromQL alarm has no separate
    // threshold, comparison operator, statistic, period, or evaluation periods.
    //
    // Note that the Kotlin SDK spells this AlarmPromQlCriteria, with a lowercase
 l in
    // "Ql". Every other AWS SDK spells it PromQL, so don't be thrown by the
 difference
    // when comparing this example against the other language versions.
    val promQlCriteria =
        AlarmPromQlCriteria {
            query = queryVal
            pendingPeriod = pendingPeriodVal
            recoveryPeriod = recoveryPeriodVal
        }

    // EvaluationCriteria is a union and is mutually exclusive with the classic
    // metricName and metrics parameters. When you use it, you must also set
    // evaluationInterval.
    val request =
        PutMetricAlarmRequest {
            alarmName = alarmNameVal
            alarmDescription = "A PromQL alarm created by the Kotlin SDK"
            evaluationCriteria =
 EvaluationCriteria.PromQlCriteria(promQlCriteria)
            evaluationInterval = evaluationIntervalVal
        }
```

```
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created PromQL alarm $alarmNameVal for query
 $queryVal")
    }
}

suspend fun describeAlarmContributors(alarmNameVal: String):
 List<AlarmContributor> {
    val contributors = mutableListOf<AlarmContributor>()

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        var token: String? = null
        do {
            val response =
                cwClient.describeAlarmContributors(
                    DescribeAlarmContributorsRequest {
                        alarmName = alarmNameVal
                        nextToken = token
                    },
                )

            response.alarmContributors?.let { contributors.addAll(it) }
            token = response.nextToken
        } while (token != null)

        if (contributors.isEmpty()) {
            println(
                "No contributors yet. The query matched no series, which usually
 means no " +
                    "OTel metrics with these labels have arrived",
            )
        }

        contributors.forEach { contributor ->
            val labels =
                contributor.contributorAttributes
                    ?.entries
                    ?.sortedBy { it.key }
                    ?.joinToString(", ") { "${it.key}=${it.value}" }
            println("${contributor.contributorId}: $labels")
            println("  reason: ${contributor.stateReason}")
        }
```

```
    }
    return contributors
}

suspend fun putAlarmMuteRule(
    muteRuleName: String,
    expressionVal: String,
    durationVal: String,
    alarmNamesVal: List<String>,
    timezoneVal: String = "America/Los_Angeles",
) {
    // For a recurring window, use a five-field cron expression,
    // cron(Minutes Hours Day-of-month Month Day-of-week), such as cron(0 2 * *
 SUN).
    // Note that this is five fields, not the six that Amazon EventBridge uses.
 For a
    // one-time window, use at(yyyy-MM-ddThh:mm), such as at(2026-09-05T02:00).
 The
    // duration is in ISO 8601 duration format, from PT1M (one minute) to P15D
 (15 days).
    val scheduleOb =
        Schedule {
            expression = expressionVal
            duration = durationVal
            timezone = timezoneVal
        }

    val request =
        PutAlarmMuteRuleRequest {
            name = muteRuleName
            description = "A mute rule created by the Kotlin SDK"
            rule =
                Rule {
                    schedule = scheduleOb
                }
            // Target up to 100 alarms. If muteTargets is omitted, the rule
 applies to
            // every alarm in the account.
            if (alarmNamesVal.isNotEmpty()) {
                muteTargets =
                    MuteTargets {
                        alarmNames = alarmNamesVal
                    }
            }
```

```
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putAlarmMuteRule(request)
        println("Successfully put alarm mute rule $muteRuleName")
    }
}

suspend fun getAlarmMuteRule(muteRuleName: String): GetAlarmMuteRuleResponse {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response =
            cwClient.getAlarmMuteRule(
                GetAlarmMuteRuleRequest {
                    alarmMuteRuleName = muteRuleName
                },
            )

        println("Mute rule ${response.name} is ${response.status?.value}")
        println("  ARN: ${response.alarmMuteRuleArn}")
        println("  schedule: ${response.rule?.schedule?.expression} for
 ${response.rule?.schedule?.duration}")
        response.muteTargets?.alarmNames?.let { println("  muted alarms:
 ${it.joinToString(", ")}") }
        return response
    }
}

suspend fun listAlarmMuteRules(alarmNameVal: String? = null):
 List<AlarmMuteRuleSummary> {
    val summaries = mutableListOf<AlarmMuteRuleSummary>()

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        var token: String? = null
        do {
            val response =
                cwClient.listAlarmMuteRules(
                    ListAlarmMuteRulesRequest {
                        alarmName = alarmNameVal
                        nextToken = token
                    },
                )

            response.alarmMuteRuleSummaries?.let { summaries.addAll(it) }
            token = response.nextToken
```

```
        } while (token != null)

        summaries.forEach { summary ->
            println("${summary.alarmMuteRuleArn} (${summary.status?.value})")
        }
    }
    return summaries
}

suspend fun deleteAlarmMuteRule(muteRuleName: String) {
    val request =
        DeleteAlarmMuteRuleRequest {
            alarmMuteRuleName = muteRuleName
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarmMuteRule(request)
        println("Successfully deleted alarm mute rule $muteRuleName")
    }
}

suspend fun stopOTelEnrichment() {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.stopOTelEnrichment(StopOTelEnrichmentRequest {})
        println("Successfully stopped OTel enrichment for this account")
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetDashboard
  - GetMetricStatistics
  - GetOTelEnrichment
  - ListAlarmMuteRules

- ListDashboards
- ListMetrics
- PutAlarmMuteRule
- PutDashboard
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating the CloudWatch OpenTelemetry experience.

```
from collections import Counter
from datetime import datetime, timedelta, timezone
import json
import logging
import random
import sys
import boto3
from botocore.exceptions import ClientError

from cloudwatch_basics import CloudWatchWrapper
from cloudwatch_otel import CloudWatchOTelWrapper

logger = logging.getLogger(__name__)

DEFAULT_QUERY = "avg by (host) (system_cpu_utilization) > 80"

# Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to 3600
 seconds.
EVALUATION_INTERVAL = 60
```

```
PENDING_PERIOD = 300
RECOVERY_PERIOD = 120

DASHES = "-" * 80


class CloudWatchScenario:
    """Runs an interactive scenario that shows how to use Amazon CloudWatch."""

    def __init__(self, cloudwatch_wrapper, otel_wrapper):
        """
        :param cloudwatch_wrapper: An object that wraps CloudWatch metric,
 statistic,
                                   and dashboard actions.
        :param otel_wrapper: An object that wraps CloudWatch OTel enrichment,
 PromQL
                             alarm, and alarm mute rule actions.
        """
        self.cloudwatch_wrapper = cloudwatch_wrapper
        self.otel_wrapper = otel_wrapper

        # Suffix the resource names so repeated runs do not collide.
        suffix = random.randint(1000, 9999)
        self.alarm_name = f"doc-example-promql-alarm-{suffix}"
        self.dashboard_name = f"doc-example-dashboard-{suffix}"
        self.mute_rule_name = f"doc-example-mute-rule-{suffix}"

        # Tracks whether this run turned enrichment on, so that cleanup only
 turns off
        # enrichment that this run started.
        self.started_enrichment = False
        self.dashboard_created = False

    def run_scenario(self):
        """Runs the eight steps of the scenario in order."""
        print(DASHES)
        print("Welcome to the Amazon CloudWatch Basics scenario.")
        print(
            "\nCloudWatch now ingests OpenTelemetry metrics natively. This
 scenario walks\n"
            "through that experience: it turns on OTel enrichment so CloudWatch
 can\n"
            "correlate incoming OTLP metrics with the resources that produced
 them, alarms\n"
```

```
            "on those metrics with a PromQL query, and shows you which individual
 series\n"
            "drove the alarm.\n"
            "\nA PromQL alarm works differently from a classic metric alarm.
 Rather than\n"
            "watching one metric and counting breaching periods, it evaluates a
 query that\n"
            "can match many series at once, and tracks each one separately as a
 contributor."
        )
        print(DASHES)

        namespaces = self.list_metrics_and_namespaces()
        self.start_otel_enrichment()
        self.explain_otlp_ingestion()
        self.create_promql_alarm()
        self.inspect_alarm_contributors()
        self.get_statistics_and_chart_metric(namespaces)
        self.mute_alarm_for_maintenance()

    def list_metrics_and_namespaces(self):
        """
        Lists the metrics and namespaces already present in the account, to
 orient the
        reader before any configuration happens.

        :return: A Counter of namespace to metric count, most common first.
        """
        print("1. List metrics and namespaces")
        print(
            "\nBefore configuring anything, let's see what CloudWatch is already
\n"
            "collecting in this account by calling ListMetrics.\n"
        )

        namespaces = Counter()
        metric_count = 0
        for metric in self.cloudwatch_wrapper.list_all_metrics():
            namespaces[metric.namespace] += 1
            metric_count += 1
            # This account may have a very large number of metrics, so stop once
 we
            # have enough to give the reader a sense of what is there.
            if metric_count >= 500:
```

```
                break

        print(f"\tFound {metric_count} metrics across {len(namespaces)}
namespaces:")
        for namespace, count in namespaces.most_common(10):
            print(f"\t  {namespace} ({count} metrics)")

        if not namespaces:
            print(
                "\tNo metrics found in this account. The statistics and dashboard
steps\n"
                "\tlater on need an existing metric, so they will be skipped."
            )

        print(DASHES)
        return namespaces

    def start_otel_enrichment(self):
        """
        Starts OTel enrichment, but only if it is not already running. Enrichment
is
        what makes CloudWatch attach AWS resource context to incoming OTLP
metrics.
        """
        print("2. Start OpenTelemetry enrichment")
        print(
            "\nEnrichment is what lets CloudWatch attach AWS resource context to
the OTLP\n"
            "metrics you send it. Without it, your metrics arrive as opaque
series with no\n"
            "connection to the resources that emitted them.\n"
            "\nWe check the current state first, and only start enrichment if it
isn't\n"
            "already on.\n"
        )

        status = self.otel_wrapper.get_otel_enrichment_status()
        print(f"\tEnrichment status: {status}")

        if status != "Running":
            self.otel_wrapper.start_otel_enrichment()
            self.started_enrichment = True
            status = self.otel_wrapper.get_otel_enrichment_status()
            print(f"\tEnrichment status: {status}")
```

```
            print(
                "\n\tNote: this run started enrichment, so the cleanup step will
stop it\n"
                "\tagain."
            )
        else:
            print(
                "\n\tEnrichment was already running, so we will leave it alone.
The cleanup\n"
                "\tstep will not stop it, because other workloads in this account
may\n"
                "\tdepend on it."
            )

        print(DASHES)

    @staticmethod
    def explain_otlp_ingestion():
        """
        Explains that OTLP metric ingestion is not an AWS SDK operation. This
step makes
        no service call; naming the gap explicitly is the point.
        """
        print("3. Send OTLP metrics to CloudWatch")
        print(
            "\nThis step is not an AWS SDK operation, and that's worth being
explicit\n"
            "about. Metrics reach CloudWatch over the OTLP protocol, through the
CloudWatch\n"
            "agent, an OpenTelemetry Collector, or an ADOT SDK. There is no
PutOTelMetrics\n"
            "API to call.\n"
            "\nPoint your collector at the CloudWatch metrics endpoint, which
follows the\n"
            "pattern\n"
            "\thttps://monitoring.<region>.amazonaws.com/v1/metrics\n"
            "\nThe endpoint is HTTP/1.1 only and does not support gRPC, so use an
otlphttp\n"
            "exporter rather than otlp. The metrics endpoint signs as
'monitoring'.\n"
            "\nSee otlp_collector_config.yaml in this folder for a working
collector\n"
            "configuration."
        )
```

```
        print(DASHES)

    def create_promql_alarm(self):
        """Creates an alarm whose evaluation is a PromQL query."""
        print("4. Create a PromQL alarm")
        print(
            "\nNow we alarm on those metrics. The comparison goes inside the
query itself:\n"
            "a PromQL alarm has no separate threshold, comparison operator,
statistic, or\n"
            "period.\n"
        )

        query = (
            input(
                f"Enter a PromQL query, or press ENTER for [{DEFAULT_QUERY}]: "
            ).strip()
            or DEFAULT_QUERY
        )

        self.otel_wrapper.create_promql_alarm(
            self.alarm_name,
            query,
            EVALUATION_INTERVAL,
            pending_period=PENDING_PERIOD,
            recovery_period=RECOVERY_PERIOD,
            description="A PromQL alarm created by the Boto3 Basics scenario.",
        )

        print(f"\tCreated alarm {self.alarm_name}:")
        print(f"\t  query:              {query}")
        print(f"\t  evaluationInterval: {EVALUATION_INTERVAL} seconds")
        print(f"\t  pendingPeriod:      {PENDING_PERIOD} seconds")
        print(f"\t  recoveryPeriod:     {RECOVERY_PERIOD} seconds")
        print(
            "\n\tA PromQL alarm starts in the OK state rather than
INSUFFICIENT_DATA, which\n"
            "\tis another way it differs from a classic alarm."
        )
        print(DASHES)

    def inspect_alarm_contributors(self):
        """
```

```
        Shows which individual series the alarm's query matched. This is the step
with
        no classic-alarm equivalent.
        """
        print("5. Inspect the alarm's contributors")
        print(
            "\nEach contributor is one series the query matched, identified by
its label\n"
            "set. This is how you find out which host is unhealthy rather than
only that\n"
            "something is. Classic alarms have no equivalent.\n"
        )

        contributors =
self.otel_wrapper.describe_alarm_contributors(self.alarm_name)

        if not contributors:
            print(
                "\tNo contributors yet. The query matched no series, which
usually means no\n"
                "\tOTel metrics with these labels have arrived. Once your
collector is\n"
                "\tsending data, each matching series appears here with its
labels and the\n"
                "\treason it breached."
            )
        else:
            print(f"\tFound {len(contributors)} contributors:")
            for contributor in contributors:
                labels = ", ".join(
                    f"{key}={value}"
                    for key, value in sorted(
                        contributor.get("ContributorAttributes", {}).items()
                    )
                )
                print(f"\t  {contributor['ContributorId']}: {labels}")
                print(f"\t    reason: {contributor.get('StateReason')}")

        print(DASHES)

    def get_statistics_and_chart_metric(self, namespaces):
        """
        Gets statistics for an existing metric and charts it on a dashboard, so
the
```

```
        reader can see what the alarm is evaluating.

        :param namespaces: The Counter of namespaces discovered in step 1.
        """
        print("6. Get statistics and chart the metric on a dashboard")
        print(
            "\nStatistics and dashboards are how you see what the alarm is
evaluating.\n"
        )

        if not namespaces:
            print("\tSkipping statistics and dashboard because no metrics exist
yet.")
            print(DASHES)
            return

        namespace = namespaces.most_common(1)[0][0]
        metric = next(
            (
                candidate
                for candidate in self.cloudwatch_wrapper.list_all_metrics()
                if candidate.namespace == namespace
            ),
            None,
        )

        if metric is None:
            print(f"\tNo metrics found in namespace {namespace}, skipping.")
            print(DASHES)
            return

        try:
            stats = self.cloudwatch_wrapper.get_metric_statistics(
                metric.namespace,
                metric.name,
                datetime.now(timezone.utc) - timedelta(days=1),
                datetime.now(timezone.utc),
                3600,
                ["Average", "Maximum"],
            )
            datapoints = stats["Datapoints"]
            print(
                f"\tStatistics for {metric.namespace} {metric.name} over the last
day:"
```

```
            )
            print(f"\t  Datapoints: {len(datapoints)}")
            for datapoint in datapoints[:3]:
                print(
                    f"\t  {datapoint['Timestamp']} average
{datapoint.get('Average')}, "
                    f"maximum {datapoint.get('Maximum')}"
                )
        except ClientError as error:
            print(f"\tCould not get statistics: {error}")

        try:
            region = self.otel_wrapper.cloudwatch_client.meta.region_name
            body = self.build_dashboard_body(metric, region)
            messages = self.cloudwatch_wrapper.put_dashboard(self.dashboard_name,
body)
            self.dashboard_created = True
            for message in messages:
                print(f"\tDashboard validation message:
{message.get('Message')}")
            print(f"\tCreated dashboard {self.dashboard_name}.")

            stored = self.cloudwatch_wrapper.get_dashboard(self.dashboard_name)
            print(
                f"\tRead the dashboard back, {len(stored)} characters of widget
JSON."
            )
        except ClientError as error:
            print(f"\tCould not create the dashboard: {error}")

        print(DASHES)

    @staticmethod
    def build_dashboard_body(metric, region):
        """
        Builds a single-widget dashboard body that charts the given metric.

        :param metric: A Boto3 CloudWatch Metric resource.
        :param region: The region the metric is in. A metric widget must name its
                       region, because a dashboard can chart metrics from
several.
        :return: The dashboard body, as a JSON string.
        """
        metric_spec = [metric.namespace, metric.name]
```

```
        for dimension in metric.dimensions or []:
            metric_spec.extend([dimension["Name"], dimension["Value"]])

        return json.dumps(
            {
                "widgets": [
                    {
                        "type": "text",
                        "x": 0,
                        "y": 0,
                        "width": 24,
                        "height": 2,
                        "properties": {
                            "markdown": "This dashboard was created
programmatically "
                            "by an AWS SDK code example."
                        },
                    },
                    {
                        "type": "metric",
                        "x": 0,
                        "y": 2,
                        "width": 12,
                        "height": 6,
                        "properties": {
                            "metrics": [metric_spec],
                            "view": "timeSeries",
                            "stat": "Average",
                            "period": 300,
                            "region": region,
                            "title": metric.name,
                        },
                    },
                ]
            }
        )

    def mute_alarm_for_maintenance(self):
        """
        Creates a mute rule so the alarm's actions are suppressed during a
maintenance
        window, then reads it back and finds it in the account's rules.
        """
        print("7. Mute the alarm for a maintenance window")
```

```
        print(
            "\nWhile a mute rule is active the targeted alarms keep evaluating
and keep\n"
            "changing state, but their actions do not fire. This is the supported
way to\n"
            "suppress notifications during planned maintenance, instead of
disabling alarm\n"
            "actions and hoping someone remembers to turn them back on.\n"
        )

        # The expression is a five-field cron expression,
        # cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
five
        # fields, not the six that Amazon EventBridge uses. For a one-time
window, use
        # at(yyyy-MM-ddThh:mm), with no seconds. The duration is an ISO 8601
duration
        # from PT1M to P15D, so PT2H rather than 2h.
        expression = "cron(0 2 * * SUN)"
        duration = "PT2H"
        tz = "America/Los_Angeles"

        self.otel_wrapper.put_alarm_mute_rule(
            self.mute_rule_name,
            expression,
            duration,
            alarm_names=[self.alarm_name],
            timezone=tz,
            description="A mute rule created by the Boto3 Basics scenario.",
        )

        print(f"\tCreated mute rule {self.mute_rule_name}:")
        print(f"\t  schedule: {expression} for {duration}")
        print(f"\t  timezone: {tz}")
        print(f"\t  targets:  {self.alarm_name}")
        print(
            "\n\tNote the two formats here. The expression is a five-field cron
expression,\n"
            "\tfive rather than the six Amazon EventBridge uses. The duration is
an ISO 8601\n"
            "\tduration, so 'PT2H' and not '2h'.\n"
            "\n\tAlso note that MuteTargets is set explicitly. If you leave it
out, the rule\n"
            "\tapplies to every alarm in the account."
```

```
        )

        mute_rule = self.otel_wrapper.get_alarm_mute_rule(self.mute_rule_name)
        print(
            f"\tRead the rule back: status {mute_rule.get('Status')}, "
            f"mute type {mute_rule.get('MuteType')}."
        )

        summaries =
self.otel_wrapper.list_alarm_mute_rules(alarm_name=self.alarm_name)
        print(f"\tFound {len(summaries)} mute rules targeting this alarm.")
        # Mute rule summaries carry no name field, only an ARN, so match on the
ARN
        # suffix.
        for summary in summaries:
            arn = summary.get("AlarmMuteRuleArn", "")
            if arn.endswith(f"/{self.mute_rule_name}") or arn.endswith(
                f":{self.mute_rule_name}"
            ):
                print(f"\t  matched by ARN: {arn} ({summary.get('Status')})")
                break

        print(DASHES)

    def clean_up(self):
        """
        Deletes the resources the scenario created. Each deletion is attempted
        independently so that one failure does not leave the remaining resources
behind.
        """
        print("8. Clean up")
        answer = input("Delete the resources this scenario created? (y/n) ")
        if answer.strip().lower() != "y":
            print(
                "\tSkipping cleanup. Note that the alarm, dashboard, and mute
rule are\n"
                "\tstill in your account, and enrichment may still be running."
            )
            print(DASHES)
            return

        try:
            self.otel_wrapper.delete_alarm_mute_rule(self.mute_rule_name)
            print(f"\tDeleted mute rule {self.mute_rule_name}.")
```

```
        except ClientError as error:
            print(f"\tCould not delete the mute rule: {error}")

        try:
            self.otel_wrapper.delete_alarms([self.alarm_name])
            print(f"\tDeleted alarm {self.alarm_name}.")
        except ClientError as error:
            print(f"\tCould not delete the alarm: {error}")

        if self.dashboard_created:
            try:
                self.cloudwatch_wrapper.delete_dashboards([self.dashboard_name])
                print(f"\tDeleted dashboard {self.dashboard_name}.")
            except ClientError as error:
                print(f"\tCould not delete the dashboard: {error}")

        if self.started_enrichment:
            try:
                self.otel_wrapper.stop_otel_enrichment()
                print("\tStopped OTel enrichment, because this run started it.")
            except ClientError as error:
                print(f"\tCould not stop OTel enrichment: {error}")
        else:
            print(
                "\tLeft OTel enrichment running, because it was already on before
 this run."
            )

        print(DASHES)


def main():
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    scenario = CloudWatchScenario(
        CloudWatchWrapper(boto3.resource("cloudwatch")),
        CloudWatchOTelWrapper(boto3.client("cloudwatch")),
    )
    try:
        scenario.run_scenario()
    except Exception:  # pylint: disable=broad-except
        logging.exception("Something went wrong with the scenario.")
    finally:
        scenario.clean_up()
```

```
    print("This concludes the Amazon CloudWatch Basics scenario.")


if __name__ == "__main__":
    sys.exit(main())
```

The class that wraps the CloudWatch OpenTelemetry operations the scenario calls.

```
import logging
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))
```

```
    def get_otel_enrichment_status(self):
        """
        Gets the current OTel enrichment status for the account.

        :return: The status, either 'Running' or 'Stopped'.
        """
        try:
            response = self.cloudwatch_client.get_o_tel_enrichment()
        except ClientError:
            logger.exception("Couldn't get the OTel enrichment status.")
            raise
        else:
            status = response["Status"]
            logger.info("OTel enrichment status is %s.", status)
            return status


    def start_otel_enrichment(self):
        """
        Turns on OTel enrichment for the account. Once enrichment is running,
        CloudWatch vended metrics that carry a resource identifier dimension,
such as
        the EC2 CPUUtilization metric with its InstanceId dimension, are
decorated with
        resource ARN and resource tag labels and become queryable with PromQL.

        Resource tags on telemetry must already be enabled for the account before
you
        call this operation.
        """
        try:
            # Boto3 splits the OTel prefix when it converts the
StartOTelEnrichment
            # operation name to snake case, so the method is
start_o_tel_enrichment.
            self.cloudwatch_client.start_o_tel_enrichment()
            logger.info("Started OTel enrichment for this account.")
        except ClientError:
            logger.exception("Couldn't start OTel enrichment.")
            raise


    def create_promql_alarm(
        self,
```

```
        alarm_name,
        query,
        evaluation_interval,
        pending_period=300,
        recovery_period=120,
        description=None,
        alarm_actions=None,
    ):
        """
        Creates an alarm that evaluates a PromQL query.

        A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
        match many series at once, and each matching series is tracked separately
as a
        *contributor*. Instead of counting breaching periods, you specify
durations: a
        contributor moves to ALARM after it breaches continuously for the pending
        period, and back to OK after it stops breaching for the recovery period.
A
        PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA.

        The PromQL evaluation parameters live in the EvaluationCriteria union,
which is
        mutually exclusive with the classic MetricName and Metrics parameters.
When you
        use EvaluationCriteria you must also set EvaluationInterval, and you must
not
        set Period, Statistic, Threshold, ComparisonOperator, EvaluationPeriods,
        DatapointsToAlarm, or TreatMissingData.

        :param alarm_name: The name of the alarm. Must be unique within the
Region.
        :param query: The PromQL query to evaluate, such as
                      'avg(cpu_utilization_percent) > 80'. The comparison belongs
in
                      the query itself; there is no separate threshold parameter.
        :param evaluation_interval: How often, in seconds, to run the query.
Valid
                                    values are 10, 20, 30, and any multiple of
60, up
                                    to 3600.
        :param pending_period: How long, in seconds, a contributor must breach
                               continuously before it moves to ALARM.
```

```
        :param recovery_period: How long, in seconds, a contributor must stop
breaching
                                before it moves back to OK.
        :param description: The description of the alarm.
        :param alarm_actions: A list of ARNs to notify when the alarm fires, such
as an
                              Amazon SNS topic.
        """
        promql_criteria = {
            "Query": query,
            "PendingPeriod": pending_period,
            "RecoveryPeriod": recovery_period,
        }
        kwargs = {
            "AlarmName": alarm_name,
            "EvaluationCriteria": {"PromQLCriteria": promql_criteria},
            "EvaluationInterval": evaluation_interval,
        }
        if description is not None:
            kwargs["AlarmDescription"] = description
        if alarm_actions is not None:
            kwargs["AlarmActions"] = alarm_actions

        try:
            self.cloudwatch_client.put_metric_alarm(**kwargs)
            logger.info("Created PromQL alarm %s for query %s.", alarm_name,
query)
        except ClientError:
            logger.exception("Couldn't create PromQL alarm %s.", alarm_name)
            raise


    def describe_alarm_contributors(self, alarm_name):
        """
        Gets the contributors for a PromQL alarm. Each contributor is one series
that
        the alarm's query matched, identified by its label set. This is how you
find out
        *which* hosts, services, or pods are breaching, rather than only that
something
        is.

        :param alarm_name: The name of the PromQL alarm.
```

```
        :return: The list of contributors. Each contributor has a ContributorId,
a
                 ContributorAttributes map of the labels that identify the
series, a
                 StateReason, and the time it last changed state.
        """
        contributors = []
        try:
            next_token = None
            while True:
                kwargs = {"AlarmName": alarm_name}
                if next_token is not None:
                    kwargs["NextToken"] = next_token
                response =
self.cloudwatch_client.describe_alarm_contributors(**kwargs)
                contributors.extend(response["AlarmContributors"])
                next_token = response.get("NextToken")
                if not next_token:
                    break
        except ClientError:
            logger.exception("Couldn't get contributors for alarm %s.",
alarm_name)
            raise
        else:
            logger.info(
                "Got %s contributors for alarm %s.", len(contributors),
alarm_name
            )
            return contributors


    def put_alarm_mute_rule(
        self,
        name,
        expression,
        duration,
        alarm_names=None,
        timezone=None,
        description=None,
    ):
        """
        Creates or updates an alarm mute rule. While a mute rule is active the
targeted
        alarms keep evaluating and keep transitioning between states, but their
```

```
        configured actions do not fire. This is the supported way to suppress
        notifications during a known maintenance window instead of disabling
 alarm
        actions and hoping someone remembers to turn them back on.

        :param name: The name of the mute rule.
        :param expression: When the rule activates. For a recurring window, use a
                           five-field cron expression,
                           'cron(Minutes Hours Day-of-month Month Day-of-week)',
 such as
                           'cron(0 2 * * SUN)' for every Sunday at 2:00 AM. Note
 that
                           this is five fields, not the six that Amazon
 EventBridge
                           uses. For a one-time window, use 'at(yyyy-MM-
ddThh:mm)',
                           such as 'at(2026-09-05T02:00)'.
        :param duration: How long the mute window lasts once it activates, in ISO
 8601
                         duration format, from 'PT1M' (one minute) to 'P15D' (15
 days).
                         For example, 'PT2H' is two hours and 'P2DT12H' is two
 days and
                         12 hours.
        :param alarm_names: The names of up to 100 alarms to mute. If omitted,
 the rule
                            applies to all alarms in the account.
        :param timezone: The time zone the expression is evaluated in, such as
                         'America/Los_Angeles'.
        :param description: The description of the mute rule.
        """
        schedule = {"Expression": expression, "Duration": duration}
        if timezone is not None:
            schedule["Timezone"] = timezone

        kwargs = {"Name": name, "Rule": {"Schedule": schedule}}
        if alarm_names is not None:
            kwargs["MuteTargets"] = {"AlarmNames": alarm_names}
        if description is not None:
            kwargs["Description"] = description

        try:
            self.cloudwatch_client.put_alarm_mute_rule(**kwargs)
            logger.info("Put alarm mute rule %s.", name)
```

```
        except ClientError:
            logger.exception("Couldn't put alarm mute rule %s.", name)
            raise


    def get_alarm_mute_rule(self, name):
        """
        Gets the full configuration of an alarm mute rule, including its
schedule, the
        alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.

        :param name: The name of the mute rule.
        :return: The mute rule.
        """
        try:
            response = self.cloudwatch_client.get_alarm_mute_rule(
                AlarmMuteRuleName=name
            )
        except ClientError:
            logger.exception("Couldn't get alarm mute rule %s.", name)
            raise
        else:
            logger.info("Got alarm mute rule %s.", name)
            return response


    def list_alarm_mute_rules(self, alarm_name=None, statuses=None):
        """
        Lists alarm mute rules in the account.

        :param alarm_name: When specified, only rules that target this alarm are
                           returned.
        :param statuses: When specified, only rules in these statuses are
returned.
                         Valid values are 'SCHEDULED', 'ACTIVE', and 'EXPIRED'.
        :return: The list of mute rule summaries.
        """
        summaries = []
        try:
            next_token = None
            while True:
                kwargs = {}
                if alarm_name is not None:
```

```
                    kwargs["AlarmName"] = alarm_name
                if statuses is not None:
                    kwargs["Statuses"] = statuses
                if next_token is not None:
                    kwargs["NextToken"] = next_token
                response = self.cloudwatch_client.list_alarm_mute_rules(**kwargs)
                summaries.extend(response.get("AlarmMuteRuleSummaries", []))
                next_token = response.get("NextToken")
                if not next_token:
                    break
        except ClientError:
            logger.exception("Couldn't list alarm mute rules.")
            raise
        else:
            logger.info("Got %s alarm mute rules.", len(summaries))
            return summaries


    def delete_alarm_mute_rule(self, name):
        """
        Deletes an alarm mute rule.

        :param name: The name of the mute rule.
        """
        try:
            self.cloudwatch_client.delete_alarm_mute_rule(AlarmMuteRuleName=name)
            logger.info("Deleted alarm mute rule %s.", name)
        except ClientError:
            logger.exception("Couldn't delete alarm mute rule %s.", name)
            raise


    def stop_otel_enrichment(self):
        """
        Turns off OTel enrichment for the account. Existing PromQL alarms are not
        deleted, but vended metrics stop being enriched with resource ARN and tag
        labels, so queries that select on those labels stop matching.
        """
        try:
            self.cloudwatch_client.stop_o_tel_enrichment()
            logger.info("Stopped OTel enrichment for this account.")
        except ClientError:
            logger.exception("Couldn't stop OTel enrichment.")
            raise
```

The metric, statistic, and dashboard operations the scenario calls.

```
   def list_all_metrics(self):
       """
       Gets every metric in the account, without filtering by namespace or name.
Use
       this to discover what CloudWatch is already collecting before you
configure
       anything.

       :return: An iterator that yields the retrieved metrics.
       """
       try:
           metric_iter = self.cloudwatch_resource.metrics.all()
           logger.info("Got all metrics for the account.")
       except ClientError:
           logger.exception("Couldn't get metrics for the account.")
           raise
       else:
           return metric_iter


   def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
       """
       Gets statistics for a metric within a specified time span. Metrics are
grouped
       into the specified period.

       :param namespace: The namespace of the metric.
       :param name: The name of the metric.
       :param start: The UTC start time of the time span to retrieve.
       :param end: The UTC end time of the time span to retrieve.
       :param period: The period, in seconds, in which to group metrics. The
period
                      must match the granularity of the metric, which depends on
                      the metric's age. For example, metrics that are older than
                      three hours have a one-minute granularity, so the period
must
                      be at least 60 and must be a multiple of 60.
```

```
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats


    def put_dashboard(self, name, body):
        """
        Creates or replaces a dashboard. The body is a JSON document describing
the
        dashboard's widgets.

        :param name: The name of the dashboard.
        :param body: The dashboard body, as a JSON string.
        :return: Any validation messages the service returned. An empty list
means the
                 dashboard body was accepted as written.
        """
        try:
            response = self.cloudwatch_resource.meta.client.put_dashboard(
                DashboardName=name, DashboardBody=body
            )
            logger.info("Put dashboard %s.", name)
        except ClientError:
            logger.exception("Couldn't put dashboard %s.", name)
            raise
        else:
```

```
            return response.get("DashboardValidationMessages", [])


    def get_dashboard(self, name):
        """
        Gets a dashboard's body, so you can confirm what the service actually
stored.

        :param name: The name of the dashboard.
        :return: The dashboard body, as a JSON string.
        """
        try:
            response = self.cloudwatch_resource.meta.client.get_dashboard(
                DashboardName=name
            )
            logger.info("Got dashboard %s.", name)
        except ClientError:
            logger.exception("Couldn't get dashboard %s.", name)
            raise
        else:
            return response["DashboardBody"]


    def delete_dashboards(self, names):
        """
        Deletes the specified dashboards.

        :param names: The names of the dashboards to delete.
        """
        try:

self.cloudwatch_resource.meta.client.delete_dashboards(DashboardNames=names)
            logger.info("Deleted dashboards %s.", ", ".join(names))
        except ClientError:
            logger.exception("Couldn't delete dashboards %s.", ", ".join(names))
            raise
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms

- DeleteDashboards
- DescribeAlarmContributors
- GetAlarmMuteRule
- GetDashboard
- GetMetricStatistics
- GetOTelEnrichment
- ListAlarmMuteRules
- ListDashboards
- ListMetrics
- PutAlarmMuteRule
- PutDashboard
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating the CloudWatch OpenTelemetry experience.

```
DASHES = ('-' * 80).freeze
DEFAULT_QUERY = 'avg by (host) (system_cpu_utilization) > 80'.freeze

# Valid evaluation intervals are 10, 20, 30, or any multiple of 60 up to 3600
 seconds.
EVALUATION_INTERVAL = 60
PENDING_PERIOD = 300
RECOVERY_PERIOD = 120
```

```
# Lists the metrics and namespaces already present in the account, to orient the
 reader
# before any configuration happens.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Hash] A hash of namespace to the metrics found in it, most populated
 first.
def metrics_by_namespace(cloudwatch_client)
  by_namespace = {}
  metric_count = 0

  cloudwatch_client.list_metrics.each_page do |page|
    page.metrics.each do |metric|
      (by_namespace[metric.namespace] ||= []) << metric
      metric_count += 1
    end
    # This account may have a very large number of metrics, so stop once we have
 enough
    # to give the reader a sense of what is there.
    break if metric_count >= 500
  end

  puts "\tFound #{metric_count} metrics across #{by_namespace.size} namespaces:"
  by_namespace
    .sort_by { |_namespace, metrics| -metrics.size }
    .first(10)
    .each { |namespace, metrics| puts "\t  #{namespace} (#{metrics.size}
 metrics)" }

  if by_namespace.empty?
    puts "\tNo metrics found in this account. The statistics and dashboard steps
 later"
    puts "\ton need an existing metric, so they will be skipped."
  end

  by_namespace
rescue StandardError => e
  puts "Error listing metrics: #{e.message}"
  {}
end

# Turns on OTel enrichment, but only if it is not already running. Enrichment is
 an
```

```
# account-wide setting, so this scenario only turns it off again if it was the
 thing that
# turned it on.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Boolean] true if this run started enrichment; otherwise, false.
def enrichment_started_by_example?(cloudwatch_client)
  # The Ruby SDK renders the OTel prefix as +o_tel+, so the methods are
  # +get_o_tel_enrichment+ and +start_o_tel_enrichment+.
  status = cloudwatch_client.get_o_tel_enrichment.status
  puts "\tEnrichment status: #{status}"

  if status == 'Running'
    puts
    puts "\tEnrichment was already running, so we will leave it alone. The
 cleanup step"
    puts "\twill not stop it, because other workloads in this account may depend
 on it."
    return false
  end

  cloudwatch_client.start_o_tel_enrichment
  puts "\tEnrichment status: #{cloudwatch_client.get_o_tel_enrichment.status}"
  puts
  puts "\tNote: this run started enrichment, so the cleanup step will stop it
 again."
  true
rescue StandardError => e
  puts "Error starting OTel enrichment: #{e.message}"
  false
end

# Creates an alarm whose evaluation is a PromQL query.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the alarm to create.
# @param query [String] The PromQL query to evaluate.
# @return [Boolean] true if the alarm was created; otherwise, false.
def promql_alarm_created?(cloudwatch_client, alarm_name, query)
  # The comparison belongs in the query itself. A PromQL alarm has no separate
 threshold,
```

```
  # comparison operator, statistic, period, or evaluation periods. Note that the
 Ruby SDK
  # spells the criteria member +prom_ql_criteria+.
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: 'A PromQL alarm created by the AWS SDK for Ruby Basics
 scenario.',
    evaluation_criteria: {
      prom_ql_criteria: {
        query: query,
        pending_period: PENDING_PERIOD,
        recovery_period: RECOVERY_PERIOD
      }
    },
    evaluation_interval: EVALUATION_INTERVAL
  )
  true
rescue StandardError => e
  puts "Error creating PromQL alarm: #{e.message}"
  false
end

# Prints the contributors to a PromQL alarm. Each contributor is one series the
 query
# matched, identified by its label set.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the PromQL alarm.
# @return [void]
def report_alarm_contributors(cloudwatch_client, alarm_name)
  contributors = []
  next_token = nil

  loop do
    response = cloudwatch_client.describe_alarm_contributors(
      alarm_name: alarm_name,
      next_token: next_token
    )
    contributors.concat(response.alarm_contributors)
    next_token = response.next_token
    # A page can come back empty while still carrying a token, so keep going
 until the
    # token itself is gone rather than stopping at the first empty page.
```

```
    break if next_token.nil? || next_token.empty?
  end

  if contributors.empty?
    puts "\tNo contributors yet. The query matched no series, which usually means
 no"
    puts "\tOTel metrics with these labels have arrived. Once your collector is
 sending"
    puts "\tdata, each matching series appears here with its labels and why it
 breached."
    return
  end

  puts "\tFound #{contributors.size} contributors:"
  contributors.each do |contributor|
    labels = contributor.contributor_attributes.sort.map { |k, v|
 "#{k}=#{v}" }.join(', ')
    puts "\t  #{contributor.contributor_id}: #{labels}"
    puts "\t    reason: #{contributor.state_reason}"
  end
rescue StandardError => e
  puts "Error describing alarm contributors: #{e.message}"
end

# Builds a single-widget dashboard body that charts the given metric.
#
# @param metric [Aws::CloudWatch::Types::Metric] The metric to chart.
# @param region [String] The region the metric is in. A metric widget must name
 its
#   region, because a dashboard can chart metrics from several.
# @return [String] The dashboard body, as JSON.
def dashboard_body(metric, region)
  metric_spec = [metric.namespace, metric.metric_name]
  metric.dimensions.each { |dimension| metric_spec.push(dimension.name,
 dimension.value) }

  {
    widgets: [
      {
        type: 'text',
        x: 0, y: 0, width: 24, height: 2,
        properties: {
          markdown: 'This dashboard was created programmatically by an AWS SDK
 code example.'
```

```
        }
      },
      {
        type: 'metric',
        x: 0, y: 2, width: 12, height: 6,
        properties: {
          metrics: [metric_spec],
          view: 'timeSeries',
          stat: 'Average',
          period: 300,
          region: region,
          title: metric.metric_name
        }
      }
    ]
  }.to_json
end

# Gets statistics for an existing metric and charts it on a dashboard, so the
 reader can
# see what the alarm is evaluating.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param dashboard_name [String] The name of the dashboard to create.
# @param by_namespace [Hash] The namespaces and metrics discovered in step 1.
# @return [Boolean] true if a dashboard was created; otherwise, false.
def chart_metric_on_dashboard(cloudwatch_client, dashboard_name, by_namespace)
  if by_namespace.empty?
    puts "\tSkipping statistics and dashboard because no metrics exist yet."
    return false
  end

  metric = by_namespace.max_by { |_namespace, metrics| metrics.size }.last.first

  stats = cloudwatch_client.get_metric_statistics(
    namespace: metric.namespace,
    metric_name: metric.metric_name,
    dimensions: metric.dimensions,
    start_time: Time.now - (60 * 60 * 24),
    end_time: Time.now,
    period: 3600,
    statistics: %w[Average Maximum]
  )
```

```
  puts "\tStatistics for #{metric.namespace} #{metric.metric_name} over the last
 day:"
  puts "\t  Datapoints: #{stats.datapoints.size}"
  stats.datapoints.first(3).each do |datapoint|
    puts "\t  #{datapoint.timestamp} average #{datapoint.average}, maximum
 #{datapoint.maximum}"
  end

  response = cloudwatch_client.put_dashboard(
    dashboard_name: dashboard_name,
    dashboard_body: dashboard_body(metric, cloudwatch_client.config.region)
  )
  response.dashboard_validation_messages.each do |message|
    puts "\tDashboard validation message: #{message.message}"
  end
  puts "\tCreated dashboard #{dashboard_name}."

  stored = cloudwatch_client.get_dashboard(dashboard_name: dashboard_name)
  puts "\tRead the dashboard back, #{stored.dashboard_body.length} characters of
 widget JSON."
  true
rescue StandardError => e
  puts "Error getting statistics or creating the dashboard: #{e.message}"
  false
end

# Creates a mute rule so the alarm's actions are suppressed during a maintenance
 window,
# then reads it back and finds it in the account's rules.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param mute_rule_name [String] The name of the mute rule to create.
# @param alarm_name [String] The name of the alarm to mute.
# @return [void]
def mute_alarm_for_maintenance(cloudwatch_client, mute_rule_name, alarm_name)
  # The expression is a five-field cron expression,
  # cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is five
 fields, not
  # the six that Amazon EventBridge uses. For a one-time window, use at(yyyy-MM-
ddThh:mm),
  # with no seconds. The duration is an ISO 8601 duration from PT1M to P15D, so
 PT2H
  # rather than 2h.
```

```
 expression = 'cron(0 2 * * SUN)'
 duration = 'PT2H'
 timezone = 'America/Los_Angeles'

 cloudwatch_client.put_alarm_mute_rule(
   name: mute_rule_name,
   description: 'A mute rule created by the AWS SDK for Ruby Basics scenario.',
   rule: {
     schedule: {
       expression: expression,
       duration: duration,
       timezone: timezone
     }
   },
   # Target up to 100 alarms. If mute_targets is omitted, the rule applies to
every alarm
   # in the account.
   mute_targets: { alarm_names: [alarm_name] }
 )

 puts "\tCreated mute rule #{mute_rule_name}:"
 puts "\t  schedule: #{expression} for #{duration}"
 puts "\t  timezone: #{timezone}"
 puts "\t  targets:  #{alarm_name}"
 puts
 puts "\tNote the two formats here. The expression is a five-field cron
expression, five"
 puts "\trather than the six Amazon EventBridge uses. The duration is an ISO
8601"
 puts "\tduration, so 'PT2H' and not '2h'."
 puts
 puts "\tAlso note that mute_targets is set explicitly. If you leave it out, the
rule"
 puts "\tapplies to every alarm in the account."

 rule = cloudwatch_client.get_alarm_mute_rule(alarm_mute_rule_name:
mute_rule_name)
 puts "\tRead the rule back: status #{rule.status}, mute type
#{rule.mute_type}."

 summaries = cloudwatch_client.list_alarm_mute_rules(alarm_name: alarm_name)
                              .alarm_mute_rule_summaries
 puts "\tFound #{summaries.size} mute rules targeting this alarm."
```

```
  # Mute rule summaries carry no name field, only an ARN, so match on the ARN
 suffix.
  match = summaries.find do |summary|
    summary.alarm_mute_rule_arn.end_with?("/#{mute_rule_name}",
 ":#{mute_rule_name}")
  end
  puts "\t  matched by ARN: #{match.alarm_mute_rule_arn} (#{match.status})" if
 match
rescue StandardError => e
  puts "Error muting the alarm: #{e.message}"
end

# Deletes the resources the scenario created. Each deletion is attempted
 independently so
# that one failure does not leave the remaining resources behind.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param names [Hash] The alarm, dashboard, and mute rule names to delete.
# @param started_here [Boolean] Whether this run turned OTel enrichment on.
# @return [void]
def clean_up(cloudwatch_client, names, started_here)
  begin
    cloudwatch_client.delete_alarm_mute_rule(alarm_mute_rule_name:
 names[:mute_rule])
    puts "\tDeleted mute rule #{names[:mute_rule]}."
  rescue StandardError => e
    puts "\tCould not delete the mute rule: #{e.message}"
  end

  begin
    cloudwatch_client.delete_alarms(alarm_names: [names[:alarm]])
    puts "\tDeleted alarm #{names[:alarm]}."
  rescue StandardError => e
    puts "\tCould not delete the alarm: #{e.message}"
  end

  if names[:dashboard]
    begin
      cloudwatch_client.delete_dashboards(dashboard_names: [names[:dashboard]])
      puts "\tDeleted dashboard #{names[:dashboard]}."
    rescue StandardError => e
      puts "\tCould not delete the dashboard: #{e.message}"
    end
```

```
  end

  unless started_here
    puts "\tLeft OTel enrichment running, because it was already on before this
 run."
    return
  end

  begin
    cloudwatch_client.stop_o_tel_enrichment
    puts "\tStopped OTel enrichment, because this run started it."
  rescue StandardError => e
    puts "\tCould not stop OTel enrichment: #{e.message}"
  end
end

# Prints the scenario's introduction.
#
# @return [void]
def print_intro
  puts DASHES
  puts 'Welcome to the Amazon CloudWatch Basics scenario.'
  puts
  puts 'CloudWatch now ingests OpenTelemetry metrics natively. This scenario
 walks through'
  puts 'that experience: it turns on OTel enrichment so CloudWatch can correlate
 incoming'
  puts 'OTLP metrics with the resources that produced them, alarms on those
 metrics with a'
  puts 'PromQL query, and shows you which individual series drove the alarm.'
  puts
  puts 'A PromQL alarm works differently from a classic metric alarm. Rather than
 watching'
  puts 'one metric and counting breaching periods, it evaluates a query that can
 match many'
  puts 'series at once, and tracks each one separately as a contributor.'
  puts DASHES
end

# Explains that OTLP metric ingestion is not an AWS SDK operation. This step
 makes no
# service call; naming the gap explicitly is the point.
#
# @return [void]
```

```
def explain_otlp_ingestion
  puts '3. Send OTLP metrics to CloudWatch'
  puts
  puts 'This step is not an AWS SDK operation, and that\'s worth being explicit
 about.'
  puts 'Metrics reach CloudWatch over the OTLP protocol, through the CloudWatch
 agent, an'
  puts 'OpenTelemetry Collector, or an ADOT SDK. There is no PutOTelMetrics API
 to call.'
  puts
  puts 'Point your collector at the CloudWatch metrics endpoint, which follows
 the pattern'
  puts "\thttps://monitoring.<region>.amazonaws.com/v1/metrics"
  puts
  puts 'The endpoint is HTTP/1.1 only and does not support gRPC, so use an
 otlphttp'
  puts 'exporter rather than otlp. The metrics endpoint signs as "monitoring".'
  puts DASHES
end

# Prompts for a PromQL query and creates an alarm that evaluates it.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the alarm to create.
# @return [void]
def create_promql_alarm_step(cloudwatch_client, alarm_name)
  puts '4. Create a PromQL alarm'
  puts
  puts 'Now we alarm on those metrics. The comparison goes inside the query
 itself: a'
  puts 'PromQL alarm has no separate threshold, comparison operator, statistic,
 or period.'
  puts
  print "Enter a PromQL query, or press ENTER for [#{DEFAULT_QUERY}]: "
  input = $stdin.gets
  query = input.nil? || input.strip.empty? ? DEFAULT_QUERY : input.strip

  if promql_alarm_created?(cloudwatch_client, alarm_name, query)
    puts "\tCreated alarm #{alarm_name}:"
    puts "\t  query:              #{query}"
    puts "\t  evaluationInterval: #{EVALUATION_INTERVAL} seconds"
    puts "\t  pendingPeriod:      #{PENDING_PERIOD} seconds"
    puts "\t  recoveryPeriod:     #{RECOVERY_PERIOD} seconds"
```

```
    puts
    puts "\tA PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA,
 which is"
    puts "\tanother way it differs from a classic alarm."
  end
  puts DASHES
end

# Runs the eight steps of the scenario in order.
def run_me
  region = 'us-east-1'
  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Suffix the resource names so repeated runs do not collide.
  suffix = rand(1000..9999)
  alarm_name = "doc-example-promql-alarm-#{suffix}"
  dashboard_name = "doc-example-dashboard-#{suffix}"
  mute_rule_name = "doc-example-mute-rule-#{suffix}"

  print_intro

  puts '1. List metrics and namespaces'
  puts
  puts 'Before configuring anything, let\'s see what CloudWatch is already
 collecting in'
  puts 'this account by calling ListMetrics.'
  puts
  by_namespace = metrics_by_namespace(cloudwatch_client)
  puts DASHES

  puts '2. Start OpenTelemetry enrichment'
  puts
  puts 'Enrichment is what lets CloudWatch attach AWS resource context to the
 OTLP metrics'
  puts 'you send it. Without it, your metrics arrive as opaque series with no
 connection to'
  puts 'the resources that emitted them.'
  puts
  puts 'We check the current state first, and only start enrichment if it isn\'t
 already on.'
  puts
  started_here = enrichment_started_by_example?(cloudwatch_client)
  puts DASHES
```

```
  explain_otlp_ingestion
  create_promql_alarm_step(cloudwatch_client, alarm_name)

  puts '5. Inspect the alarm\'s contributors'
  puts
  puts 'Each contributor is one series the query matched, identified by its label
 set. This'
  puts 'is how you find out which host is unhealthy rather than only that
 something is.'
  puts 'Classic alarms have no equivalent.'
  puts
  report_alarm_contributors(cloudwatch_client, alarm_name)
  puts DASHES

  puts '6. Get statistics and chart the metric on a dashboard'
  puts
  puts 'Statistics and dashboards are how you see what the alarm is evaluating.'
  puts
  dashboard_created = chart_metric_on_dashboard(cloudwatch_client,
 dashboard_name, by_namespace)
  puts DASHES

  mute_alarm_step(cloudwatch_client, mute_rule_name, alarm_name)

  clean_up_step(
    cloudwatch_client,
    { alarm: alarm_name, dashboard: dashboard_created ? dashboard_name : nil,
      mute_rule: mute_rule_name },
    started_here
  )

  puts 'This concludes the Amazon CloudWatch Basics scenario.'
end

# Explains what a mute rule does, then creates one for the scenario's alarm.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param mute_rule_name [String] The name of the mute rule to create.
# @param alarm_name [String] The name of the alarm to mute.
# @return [void]
def mute_alarm_step(cloudwatch_client, mute_rule_name, alarm_name)
  puts '7. Mute the alarm for a maintenance window'
  puts
```

```
  puts 'While a mute rule is active the targeted alarms keep evaluating and keep
 changing'
  puts 'state, but their actions do not fire. This is the supported way to
 suppress'
  puts 'notifications during planned maintenance, instead of disabling alarm
 actions and'
  puts 'hoping someone remembers to turn them back on.'
  puts
  mute_alarm_for_maintenance(cloudwatch_client, mute_rule_name, alarm_name)
  puts DASHES
end

# Asks whether to delete the resources the scenario created, and deletes them if
 so.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param names [Hash] The alarm, dashboard, and mute rule names to delete.
# @param started_here [Boolean] Whether this run turned OTel enrichment on.
# @return [void]
def clean_up_step(cloudwatch_client, names, started_here)
  puts '8. Clean up'
  print 'Delete the resources this scenario created? (y/n) '
  answer = $stdin.gets
  if answer.nil? || answer.strip.downcase != 'y'
    puts "\tSkipping cleanup. Note that the alarm, dashboard, and mute rule are
 still in"
    puts "\tyour account, and enrichment may still be running."
  else
    clean_up(cloudwatch_client, names, started_here)
  end
  puts DASHES
end
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DeleteDashboards
  - DescribeAlarmContributors
  - GetAlarmMuteRule

- GetDashboard
- GetMetricStatistics
- GetOTelEnrichment
- ListAlarmMuteRules
- ListDashboards
- ListMetrics
- PutAlarmMuteRule
- PutDashboard
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for CloudWatch using AWS SDKs

The following code examples demonstrate how to perform individual CloudWatch actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the CloudWatch API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for CloudWatch using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon CloudWatch API Reference.

Examples

- Use DeleteAlarmMuteRule with an AWS SDK
- Use DeleteAlarms with an AWS SDK or CLI
- Use DeleteAnomalyDetector with an AWS SDK or CLI
- Use DeleteDashboards with an AWS SDK or CLI
- Use DescribeAlarmContributors with an AWS SDK

- Use DescribeAlarmHistory with an AWS SDK or CLI
- Use DescribeAlarms with an AWS SDK or CLI
- Use DescribeAlarmsForMetric with an AWS SDK or CLI
- Use DescribeAnomalyDetectors with an AWS SDK or CLI
- Use DisableAlarmActions with an AWS SDK or CLI
- Use EnableAlarmActions with an AWS SDK or CLI
- Use GetAlarmMuteRule with an AWS SDK
- Use GetDashboard with an AWS SDK or CLI
- Use GetMetricData with an AWS SDK or CLI
- Use GetMetricStatistics with an AWS SDK or CLI
- Use GetMetricWidgetImage with an AWS SDK or CLI
- Use GetOTelEnrichment with an AWS SDK
- Use ListAlarmMuteRules with an AWS SDK
- Use ListDashboards with an AWS SDK or CLI
- Use ListMetrics with an AWS SDK or CLI
- Use PutAlarmMuteRule with an AWS SDK
- Use PutAnomalyDetector with an AWS SDK or CLI
- Use PutDashboard with an AWS SDK or CLI
- Use PutMetricAlarm with an AWS SDK or CLI
- Use PutMetricData with an AWS SDK or CLI
- Use StartOTelEnrichment with an AWS SDK
- Use StopOTelEnrichment with an AWS SDK

Use `DeleteAlarmMuteRule` with an AWS SDK

The following code examples show how to use `DeleteAlarmMuteRule`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an alarm mute rule. The alarms it targeted resume firing their
actions.
    /// </summary>
    /// <param name="name">The name of the mute rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarmMuteRule(string name)
    {
        var response = await _amazonCloudWatch.DeleteAlarmMuteRuleAsync(
            new DeleteAlarmMuteRuleRequest
            {
                AlarmMuteRuleName = name
            });

        _logger.LogInformation($"Deleted alarm mute rule {name}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmMuteRuleRequest.h>
#include <iostream>
```

Delete the alarm mute rule.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::DeleteAlarmMuteRuleRequest request;
        request.SetAlarmMuteRuleName(mute_rule_name);

        auto outcome = cw.DeleteAlarmMuteRule(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete alarm mute rule: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            std::cout << "Successfully deleted alarm mute rule " <<
mute_rule_name
                      << std::endl;
        }
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Deletes an alarm mute rule. The alarms it targeted resume firing their
 actions.
     *
     * @param cw   the CloudWatch client
     * @param name the name of the mute rule
     */
    public static void deleteAlarmMuteRule(CloudWatchClient cw, String name) {
        try {
            cw.deleteAlarmMuteRule(DeleteAlarmMuteRuleRequest.builder()
                    .alarmMuteRuleName(name)
                    .build());

            System.out.printf("Deleted alarm mute rule %s.%n", name);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DeleteAlarmMuteRuleCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Delete an alarm mute rule. The alarms it targeted resume firing their actions.
const run = async () => {
  const command = new DeleteAlarmMuteRuleCommand({
```

```
    AlarmMuteRuleName: process.env.CLOUDWATCH_MUTE_RULE_NAME, // Set
 CLOUDWATCH_MUTE_RULE_NAME to the name of an existing mute rule.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteAlarmMuteRule(muteRuleName: String) {
    val request =
        DeleteAlarmMuteRuleRequest {
            alarmMuteRuleName = muteRuleName
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAlarmMuteRule(request)
        println("Successfully deleted alarm mute rule $muteRuleName")
    }
}
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def delete_alarm_mute_rule(self, name):
        """
        Deletes an alarm mute rule.

        :param name: The name of the mute rule.
        """
        try:
            self.cloudwatch_client.delete_alarm_mute_rule(AlarmMuteRuleName=name)
            logger.info("Deleted alarm mute rule %s.", name)
        except ClientError:
```

```
            logger.exception("Couldn't delete alarm mute rule %s.", name)
            raise
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Deletes an alarm mute rule. The alarms it targeted resume firing their actions.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param name [String] The name of the mute rule.
# @return [Boolean] true if the mute rule was deleted; otherwise, false.
def alarm_mute_rule_deleted?(cloudwatch_client, name)
  cloudwatch_client.delete_alarm_mute_rule(alarm_mute_rule_name: name)
  true
rescue StandardError => e
  puts "Error deleting alarm mute rule: #{e.message}"
  false
end
```

- For API details, see DeleteAlarmMuteRule in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAlarms` with an AWS SDK or CLI

The following code examples show how to use `DeleteAlarms`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Get started with alarms
- Manage custom metrics and alarms
- Run CPU stress tests on virtual machine instances using fault injection
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see DeleteAlarms in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DeleteAlarmsRequest.h>
#include <iostream>
```

Delete the alarm.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DeleteAlarmsRequest request;
        request.AddAlarmNames(alarm_name);

        auto outcome = cw.DeleteAlarms(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to delete CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully deleted CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- For API details, see DeleteAlarms in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete an alarm

The following `delete-alarms` example deletes the Amazon CloudWatch alarm named "myalarm".

```
aws cloudwatch delete-alarms \
    --alarm-names myalarm
```

This command produces no output.

For more information, see Edit or delete a CloudWatch alarm in the *Amazon CloudWatch User Guide*.

- For API details, see DeleteAlarms in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Deletes a CloudWatch alarm.
    *
    * @param alarmName the name of the alarm to be deleted
    * @return a {@link CompletableFuture} representing the asynchronous
operation to delete the alarm
    * the {@link DeleteAlarmsResponse} is returned when the operation completes
successfully,
    * or a {@link RuntimeException} is thrown if the operation fails
    */
   public CompletableFuture<DeleteAlarmsResponse> deleteCWAlarmAsync(String
alarmName) {
```

```
        DeleteAlarmsRequest request = DeleteAlarmsRequest.builder()
            .alarmNames(alarmName)
            .build();

        return getAsyncClient().deleteAlarms(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    throw new RuntimeException("Failed to delete the alarm:{} " +
 alarmName, exception);
                } else {
                    logger.info("Successfully deleted alarm {} ", alarmName);
                }
            });
    }
```

- For API details, see DeleteAlarms in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAlarms in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteAlarms in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteAlarm(alarmNameVal: String) {
    val request =
        DeleteAlarmsRequest {
            alarmNames = listOf(alarmNameVal)
        }

    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        cwClient.deleteAlarms(request)
        println("Successfully deleted alarm $alarmNameVal")
    }
}
```

- For API details, see DeleteAlarms in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
 metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

- For API details, see DeleteAlarms in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    lo_cwt->deletealarms(
      it_alarmnames = it_alarm_names ).
    MESSAGE 'Alarms deleted.' TYPE 'I'.
  CATCH /aws1/cx_cwtresourcenotfound.
    MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
ENDTRY.
```

- For API details, see DeleteAlarms in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAnomalyDetector` with an AWS SDK or CLI

The following code examples show how to use `DeleteAnomalyDetector`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteAnomalyDetector in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete a specified anomaly detection model

The following `delete-anomaly-detector` example deletes an anomaly detector model in the specified account.

```
aws cloudwatch delete-anomaly-detector \
```

```
    --namespace AWS/Logs \
    --metric-name IncomingBytes \
    --stat SampleCount
```

This command produces no output.

For more information, see Deleting an anomaly detection model in the *Amazon CloudWatch User Guide*.

- For API details, see DeleteAnomalyDetector in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Deletes an Anomaly Detector.
    *
    * @param fileName the name of the file containing the Anomaly Detector
configuration
    * @return a CompletableFuture that represents the asynchronous deletion of
the Anomaly Detector
    */
   public CompletableFuture<DeleteAnomalyDetectorResponse>
deleteAnomalyDetectorAsync(String fileName) {
       CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               return new ObjectMapper().readTree(parser); // Return the root
node
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
e);
           }
```

```
        });

        return readFileFuture.thenCompose(rootNode -> {
            String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
            String customMetricName =
rootNode.findValue("customMetricName").asText();

            SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                .metricName(customMetricName)
                .namespace(customMetricNamespace)
                .stat("Maximum")
                .build();

            DeleteAnomalyDetectorRequest request =
DeleteAnomalyDetectorRequest.builder()
                .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                .build();

            return getAsyncClient().deleteAnomalyDetector(request);
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Failed to delete the Anomaly
Detector", exception);
            } else {
                logger.info("Successfully deleted the Anomaly Detector.");
            }
        });
    }
```

- For API details, see DeleteAnomalyDetector in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteAnomalyDetector(
    namespaceVal: String,
    metricNameVal: String,
) {
    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            namespace = namespaceVal
            metricName = metricNameVal
            stat = "Maximum"
        }

    val request =
        DeleteAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.deleteAnomalyDetector(request)
        println("Successfully deleted the anomaly detector for metric
 $metricNameVal.")
    }
}
```

- For API details, see DeleteAnomalyDetector in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDashboards` with an AWS SDK or CLI

The following code examples show how to use `DeleteDashboards`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Creating a monitoring dashboard with function name as a variable
- Using property variables in monitoring dashboards to monitor multiple serverless functions

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteDashboards in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete specified dashboards

The following `delete-dashboards` example deletes two dashboards named `Dashboard-A` and `Dashboard-B` in the specified account.

```
aws cloudwatch delete-dashboards \
    --dashboard-names Dashboard-A Dashboard-B
```

This command produces no output.

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see DeleteDashboards in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Deletes the specified dashboard.
    *
    * @param dashboardName the name of the dashboard to be deleted
    * @return a {@link CompletableFuture} representing the asynchronous
operation of deleting the dashboard
    * @throws RuntimeException if the dashboard deletion fails
    */
   public CompletableFuture<DeleteDashboardsResponse>
deleteDashboardAsync(String dashboardName) {
       DeleteDashboardsRequest dashboardsRequest =
DeleteDashboardsRequest.builder()
           .dashboardNames(dashboardName)
           .build();

       return getAsyncClient().deleteDashboards(dashboardsRequest)
           .whenComplete((response, exception) -> {
               if (exception != null) {
                   throw new RuntimeException("Failed to delete the dashboard: "
+ dashboardName, exception);
               } else {
                   logger.info("{} was successfully deleted.", dashboardName);
               }
           });
   }
```

- For API details, see DeleteDashboards in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteDashboard(dashboardName: String) {
    val dashboardsRequest =
        DeleteDashboardsRequest {
            dashboardNames = listOf(dashboardName)
        }
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        cwClient.deleteDashboards(dashboardsRequest)
        println("$dashboardName was successfully deleted.")
    }
}
```

- For API details, see DeleteDashboards in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: Deletes the specified dashboard, promoting for confirmation before proceeding. To bypass confirmation add the -Force switch to the command.

```
Remove-CWDashboard -DashboardName Dashboard1
```

- For API details, see DeleteDashboards in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Deletes the specified dashboard, promoting for confirmation before proceeding. To bypass confirmation add the -Force switch to the command.

```
Remove-CWDashboard -DashboardName Dashboard1
```

- For API details, see DeleteDashboards in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarmContributors` with an AWS SDK

The following code examples show how to use `DescribeAlarmContributors`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get the contributors for a PromQL alarm. Each contributor is one series
that the
    /// alarm's query matched, identified by its label set. This is how you find
out which
```

```
   /// hosts, services, or pods are breaching, rather than only that something
is.
   /// </summary>
   /// <param name="alarmName">The name of the PromQL alarm.</param>
   /// <returns>The list of contributors.</returns>
   public async Task<List<AlarmContributor>> DescribeAlarmContributors(string
alarmName)
   {
       var results = new List<AlarmContributor>();
       string? nextToken = null;

       do
       {
           var response = await
_amazonCloudWatch.DescribeAlarmContributorsAsync(
               new DescribeAlarmContributorsRequest
               {
                   AlarmName = alarmName,
                   NextToken = nextToken
               });

           if (response.AlarmContributors != null)
           {
               results.AddRange(response.AlarmContributors);
           }

           nextToken = response.NextToken;
       } while (!string.IsNullOrEmpty(nextToken));

       _logger.LogInformation($"Got {results.Count} contributors for alarm
{alarmName}.");
       return results;
   }
```

- For API details, see DescribeAlarmContributors in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/AlarmContributor.h>
#include <aws/monitoring/model/DescribeAlarmContributorsRequest.h>
#include <iostream>
```

Describe the contributors to a PromQL alarm.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::DescribeAlarmContributorsRequest request;
        request.SetAlarmName(alarm_name);

        // Collect every page before reporting. A page can come back empty while
 still
        // carrying a next token, so the loop must keep going until the token is
 empty
        // rather than stopping at the first empty page.
        Aws::Vector<Aws::CloudWatch::Model::AlarmContributor> contributors;
        bool failed = false;
        bool done = false;
        while (!done) {
            auto outcome = cw.DescribeAlarmContributors(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to describe alarm contributors: "
                          << outcome.GetError().GetMessage() << std::endl;
```

```
                failed = true;
                break;
            }

            const auto &page = outcome.GetResult().GetAlarmContributors();
            contributors.insert(contributors.end(), page.begin(), page.end());

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }

        if (!failed) {
            if (contributors.empty()) {
                std::cout << "No contributors yet. The query matched no series, "
                             "which usually means no OTel metrics with these
labels "
                             "have arrived."
                          << std::endl;
            }
            else {
                std::cout << "Contributors for alarm " << alarm_name << ":" <<
std::endl;
                for (const auto &contributor : contributors) {
                    std::cout << "  " << contributor.GetContributorId() << ": ";
                    bool first = true;
                    for (const auto &label :
contributor.GetContributorAttributes()) {
                        if (!first) {
                            std::cout << ", ";
                        }
                        std::cout << label.first << "=" << label.second;
                        first = false;
                    }
                    std::cout << std::endl;
                    std::cout << "    reason: " << contributor.GetStateReason()
                              << std::endl;
                }
            }
        }
```

- For API details, see DescribeAlarmContributors in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Gets the contributors for a PromQL alarm. Each contributor is one series
that the
    * alarm's query matched, identified by its label set. This is how you find
out which
    * hosts, services, or pods are breaching, rather than only that something
is.
    *
    * @param cw        the CloudWatch client
    * @param alarmName the name of the PromQL alarm
    * @return the list of contributors
    */
   public static List<AlarmContributor>
describeAlarmContributors(CloudWatchClient cw, String alarmName) {
       List<AlarmContributor> contributors = new ArrayList<>();
       try {
           String nextToken = null;
           do {
               DescribeAlarmContributorsRequest request =
DescribeAlarmContributorsRequest.builder()
                       .alarmName(alarmName)
                       .nextToken(nextToken)
                       .build();

               DescribeAlarmContributorsResponse response =
cw.describeAlarmContributors(request);
               contributors.addAll(response.alarmContributors());
               nextToken = response.nextToken();
           } while (nextToken != null && !nextToken.isEmpty());

           for (AlarmContributor contributor : contributors) {
```

```
                StringBuilder labels = new StringBuilder();
                for (Map.Entry<String, String> attribute :
contributor.contributorAttributes().entrySet()) {
                    if (labels.length() > 0) {
                        labels.append(", ");
                    }

labels.append(attribute.getKey()).append("=").append(attribute.getValue());
                }
                System.out.printf("%s: %s%n", contributor.contributorId(),
labels);
                System.out.printf("  reason: %s%n", contributor.stateReason());
            }
            return contributors;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return contributors;
        }
    }
```

- For API details, see DescribeAlarmContributors in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { DescribeAlarmContributorsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Get the contributors for a PromQL alarm. Each contributor is one series that
 the
// alarm's query matched, identified by its label set. This is how you find out
 which
```

```
// hosts, services, or pods are breaching, rather than only that something is.
const run = async () => {
  const contributors = [];
  let nextToken;

  try {
    do {
      const command = new DescribeAlarmContributorsCommand({
        AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set
 CLOUDWATCH_ALARM_NAME to the name of an existing PromQL alarm.
        NextToken: nextToken,
      });
      const response = await client.send(command);
      contributors.push(...(response.AlarmContributors ?? []));
      nextToken = response.NextToken;
    } while (nextToken);

    for (const contributor of contributors) {
      const labels = Object.entries(contributor.ContributorAttributes)
        .map(([key, value]) => `${key}=${value}`)
        .join(", ");
      console.log(`${contributor.ContributorId}: ${labels}`);
      console.log(`  reason: ${contributor.StateReason}`);
    }
    return contributors;
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see DescribeAlarmContributors in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeAlarmContributors(alarmNameVal: String):
 List<AlarmContributor> {
    val contributors = mutableListOf<AlarmContributor>()

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        var token: String? = null
        do {
            val response =
                cwClient.describeAlarmContributors(
                    DescribeAlarmContributorsRequest {
                        alarmName = alarmNameVal
                        nextToken = token
                    },
                )

            response.alarmContributors?.let { contributors.addAll(it) }
            token = response.nextToken
        } while (token != null)

        if (contributors.isEmpty()) {
            println(
                "No contributors yet. The query matched no series, which usually
 means no " +
                    "OTel metrics with these labels have arrived",
            )
        }

        contributors.forEach { contributor ->
            val labels =
                contributor.contributorAttributes
                    ?.entries
                    ?.sortedBy { it.key }
```

```
                    ?.joinToString(", ") { "${it.key}=${it.value}" }
            println("${contributor.contributorId}: $labels")
            println("  reason: ${contributor.stateReason}")
        }
    }
    return contributors
}
```

- For API details, see DescribeAlarmContributors in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))
```

```
   def describe_alarm_contributors(self, alarm_name):
       """
       Gets the contributors for a PromQL alarm. Each contributor is one series
that
       the alarm's query matched, identified by its label set. This is how you
find out
       *which* hosts, services, or pods are breaching, rather than only that
something
       is.

       :param alarm_name: The name of the PromQL alarm.
       :return: The list of contributors. Each contributor has a ContributorId,
a
                ContributorAttributes map of the labels that identify the
series, a
                StateReason, and the time it last changed state.
       """
       contributors = []
       try:
           next_token = None
           while True:
               kwargs = {"AlarmName": alarm_name}
               if next_token is not None:
                   kwargs["NextToken"] = next_token
               response =
self.cloudwatch_client.describe_alarm_contributors(**kwargs)
               contributors.extend(response["AlarmContributors"])
               next_token = response.get("NextToken")
               if not next_token:
                   break
       except ClientError:
           logger.exception("Couldn't get contributors for alarm %s.",
alarm_name)
           raise
       else:
           logger.info(
               "Got %s contributors for alarm %s.", len(contributors),
alarm_name
           )
           return contributors
```

- For API details, see DescribeAlarmContributors in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Gets the contributors for a PromQL alarm. Each contributor is one series that
 the
# alarm's query matched, identified by its label set. This is how you find out
 which
# hosts, services, or pods are breaching, rather than only that something is.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the PromQL alarm.
# @return [Array] The contributors, as Aws::CloudWatch::Types::AlarmContributor.
def alarm_contributors(cloudwatch_client, alarm_name)
  contributors = []
  next_token = nil

  loop do
    response = cloudwatch_client.describe_alarm_contributors(
      alarm_name: alarm_name,
      next_token: next_token
    )
    contributors.concat(response.alarm_contributors)
    next_token = response.next_token
    break if next_token.nil? || next_token.empty?
  end

  contributors
rescue StandardError => e
  puts "Error getting alarm contributors: #{e.message}"
  []
```

```
end
```

- For API details, see DescribeAlarmContributors in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarmHistory` with an AWS SDK or CLI

The following code examples show how to use `DescribeAlarmHistory`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string
alarmName, int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDate = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDate = DateTime.UtcNow.AddDays(-historyDays)
```

```
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- For API details, see DescribeAlarmHistory in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve history for an alarm

The following `describe-alarm-history` example retrieves history for the Amazon CloudWatch alarm named "myalarm".

```
aws cloudwatch describe-alarm-history \
    --alarm-name "myalarm" \
    --history-item-type StateUpdate
```

Output:

```
{
    "AlarmHistoryItems": [
        {
            "Timestamp": "2014-04-09T18:59:06.442Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"ALARM\",\"stateReason\":\"testing purposes\"},\"newState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were not
 greater than the threshold (70.0). The most recent datapoints: [38.958,
 40.292].\",\"stateReasonData\":{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}}",
            "HistorySummary": "Alarm updated from ALARM to OK"
```

```
        },
        {
            "Timestamp": "2014-04-09T18:59:05.805Z",
            "HistoryItemType": "StateUpdate",
            "AlarmName": "myalarm",
            "HistoryData": "{\"version\":\"1.0\",\"oldState\":{\"stateValue
\":\"OK\",\"stateReason\":\"Threshold Crossed: 2 datapoints were
 not greater than the threshold (70.0). The most recent datapoints:
 [38.839999999999996, 39.714].\",\"stateReasonData\":{\"version\":
\"1.0\",\"queryDate\":\"2014-03-11T22:45:41.569+0000\",\"startDate\":
\"2014-03-11T22:30:00.000+0000\",\"statistic\":\"Average\",\"period\":300,
\"recentDatapoints\":[38.839999999999996,39.714],\"threshold\":70.0}},\"newState
\":{\"stateValue\":\"ALARM\",\"stateReason\":\"testing purposes\"}}",
            "HistorySummary": "Alarm updated from OK to ALARM"
        }
    ]
}
```

For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

- For API details, see DescribeAlarmHistory in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Retrieves the alarm history for a given alarm name and date range.
     *
     * @param fileName the path to the JSON file containing the alarm name
     * @param date     the date to start the alarm history search (in the format
 "yyyy-MM-dd'T'HH:mm:ss'Z'")
     * @return a {@code CompletableFuture<Void>} that completes when the alarm
 history has been retrieved and processed
```

```
    */
   public CompletableFuture<Void> getAlarmHistoryAsync(String fileName, String
date) {
       CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
               return rootNode.findValue("exampleAlarmName").asText(); // Return
alarmName from the JSON file
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
e);
           }
       });

       // Use the alarm name to describe alarm history with a paginator.
       return readFileFuture.thenCompose(alarmName -> {
           try {
               Instant start = Instant.parse(date);
               Instant endDate = Instant.now();
               DescribeAlarmHistoryRequest historyRequest =
DescribeAlarmHistoryRequest.builder()
                   .startDate(start)
                   .endDate(endDate)
                   .alarmName(alarmName)
                   .historyItemType(HistoryItemType.ACTION)
                   .build();

               // Use the paginator to paginate through alarm history pages.
               DescribeAlarmHistoryPublisher historyPublisher =
getAsyncClient().describeAlarmHistoryPaginator(historyRequest);
               CompletableFuture<Void> future = historyPublisher
                   .subscribe(response ->
response.alarmHistoryItems().forEach(item -> {
                       logger.info("History summary: {}",
item.historySummary());
                       logger.info("Timestamp: {}", item.timestamp());
                   }))
                   .whenComplete((result, exception) -> {
                       if (exception != null) {
```

```
                            logger.error("Error occurred while getting alarm
history: " + exception.getMessage(), exception);
                        } else {
                            logger.info("Successfully retrieved all alarm
history.");
                        }
                    });

                // Return the future to the calling code for further handling
                return future;
            } catch (Exception e) {
                throw new RuntimeException("Failed to process alarm history", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error completing alarm history
processing", exception);
            }
        });
    }
```

- For API details, see DescribeAlarmHistory in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAlarmHistory(
    alarmNameVal: String,
    startDateVal: String,
) {
    val request =
        DescribeAlarmHistoryRequest {
            alarmName = alarmNameVal
```

```
            startDate =
                aws.smithy.kotlin.runtime.time.Instant
                    .fromIso8601(startDateVal)
            endDate =
                aws.smithy.kotlin.runtime.time.Instant
                    .now()
            historyItemType = HistoryItemType.Action
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarmHistory(request)
        val historyItems = response.alarmHistoryItems
        if (historyItems.isNullOrEmpty()) {
            println("No alarm history data found for $alarmNameVal.")
        } else {
            for (item in historyItems) {
                println("History summary: ${item.historySummary}")
                println("Time stamp: ${item.timestamp}")
            }
        }
    }
}
```

- For API details, see DescribeAlarmHistory in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarms` with an AWS SDK or CLI

The following code examples show how to use `DescribeAlarms`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Get started with alarms
- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms =
_amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- For API details, see DescribeAlarms in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list information about an alarm

The following `describe-alarms` example provides information about the alarm named "myalarm".

```
aws cloudwatch describe-alarms \
    --alarm-names "myalarm"
```

Output:

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:myalarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2012-12-27T00:49:54.032Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-west-2:123456789012:myHighCpuAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myalarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
```

```
        }
    ]
}
```

For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

- For API details, see DescribeAlarms in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Describes the CloudWatch alarms of the 'METRIC_ALARM' type.
    *
    * @return a {@link CompletableFuture} that represents the asynchronous
operation
    * of describing the CloudWatch alarms. The future completes when the
    * operation is finished, either successfully or with an error.
    */
   public CompletableFuture<Void> describeAlarmsAsync() {
       List<AlarmType> typeList = new ArrayList<>();
       typeList.add(AlarmType.METRIC_ALARM);
       DescribeAlarmsRequest alarmsRequest = DescribeAlarmsRequest.builder()
           .alarmTypes(typeList)
           .maxRecords(10)
           .build();

       return getAsyncClient().describeAlarms(alarmsRequest)
           .thenAccept(response -> {
               List<MetricAlarm> alarmList = response.metricAlarms();
               for (MetricAlarm alarm : alarmList) {
                   logger.info("Alarm name: {}", alarm.alarmName());
```

```
                    logger.info("Alarm description: {} ",
 alarm.alarmDescription());
                }
            })
            .whenComplete((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to describe alarms: {}",
 ex.getMessage());
                } else {
                    logger.info("Successfully described alarms.");
                }
            });
    }
```

- For API details, see DescribeAlarms in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun desCWAlarms() {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarms(DescribeAlarmsRequest {})
        response.metricAlarms?.forEach { alarm ->
            println("Retrieved alarm ${alarm.alarmName}")
        }
    }
}
```

- For API details, see DescribeAlarms in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns all the alarms including Composite and Metric Alarms from CloudWatch.

```
Get-CWAlarm -MaxRecords 1
```

Output:

```
CompositeAlarms MetricAlarms         NextToken
--------------- ------------         ---------
                {MetricAlarms-01}    NextToken-01
                {MetricAlarms-02}    NextToken-02
                {MetricAlarms-03}    NextToken-03
```

Example 2: Returns only the composite alarms data from CloudWatch after setting -AlarmType parameter to CompositeAlarms.

```
Get-CWAlarm -AlarmType 'CompositeAlarms'
```

Output:

```
CompositeAlarms        MetricAlarms NextToken
---------------        ------------ ---------
{CompositeAlarms-01}
{CompositeAlarms-02}
{CompositeAlarms-03}
```

- For API details, see DescribeAlarms in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-cloudwatch'

# Lists the names of available Amazon CloudWatch alarms.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   list_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def list_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
  if response.metric_alarms.count.positive?
    response.metric_alarms.each do |alarm|
      puts alarm.alarm_name
    end
  else
    puts 'No alarms found.'
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end
```

- For API details, see DescribeAlarms in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_cwt->describealarms(                " oo_result is
 returned for testing purposes. "
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarms retrieved.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
```

```
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see DescribeAlarms in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAlarmsForMetric` with an AWS SDK or CLI

The following code examples show how to use `DescribeAlarmsForMetric`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
```

```
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DescribeAlarmsRequest.h>
#include <aws/monitoring/model/DescribeAlarmsResult.h>
#include <iomanip>
#include <iostream>
```

Describe the alarms.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::DescribeAlarmsRequest request;
        request.SetMaxRecords(1);

        bool done = false;
        bool header = false;
```

```
        while (!done)
        {
            auto outcome = cw.DescribeAlarms(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to describe CloudWatch alarms:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left <<
                    std::setw(32) << "Name" <<
                    std::setw(64) << "Arn" <<
                    std::setw(64) << "Description" <<
                    std::setw(20) << "LastUpdated" <<
                    std::endl;
                header = true;
            }

            const auto &alarms = outcome.GetResult().GetMetricAlarms();
            for (const auto &alarm : alarms)
            {
                std::cout << std::left <<
                    std::setw(32) << alarm.GetAlarmName() <<
                    std::setw(64) << alarm.GetAlarmArn() <<
                    std::setw(64) << alarm.GetAlarmDescription() <<
                    std::setw(20) <<
                    alarm.GetAlarmConfigurationUpdatedTimestamp().ToGmtString(
                        SIMPLE_DATE_FORMAT_STR) <<
                    std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To display information about alarms associated with a metric

The following `describe-alarms-for-metric` example displays information about any alarms associated with the Amazon EC2 CPUUtilization metric and the instance with the ID i-0c986c72.

```
aws cloudwatch describe-alarms-for-metric \
    --metric-name CPUUtilization \
    --namespace AWS/EC2 \
    --dimensions Name=InstanceId,Value=i-0c986c72
```

Output:

```
{
    "MetricAlarms": [
        {
            "EvaluationPeriods": 10,
            "AlarmArn": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:myHighCpuAlarm2",
            "StateUpdatedTimestamp": "2013-10-30T03:03:51.479Z",
            "AlarmConfigurationUpdatedTimestamp": "2013-10-30T03:03:50.865Z",
            "ComparisonOperator": "GreaterThanOrEqualToThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-west-2:123456789012:NotifyMe"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2013-10-30T03:03:51.479+0000\",\"startDate\":\"2013-10-30T02:08:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":
[40.698,39.612,42.432,39.796,38.816,42.28,42.854,40.088,40.760000000000005,41.316],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm2",
            "Dimensions": [
                {
                    "Name": "InstanceId",
```

```
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 10 datapoints were not
 greater than or equal to the threshold (70.0). The most recent datapoints:
 [40.760000000000005, 41.316].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": true,
            "MetricName": "CPUUtilization"
        },
        {
            "EvaluationPeriods": 2,
            "AlarmArn": "arn:aws:cloudwatch:us-
west-2:123456789012:alarm:myHighCpuAlarm",
            "StateUpdatedTimestamp": "2014-04-09T18:59:06.442Z",
            "AlarmConfigurationUpdatedTimestamp": "2014-04-09T22:26:05.958Z",
            "ComparisonOperator": "GreaterThanThreshold",
            "AlarmActions": [
                "arn:aws:sns:us-west-2:123456789012:HighCPUAlarm"
            ],
            "Namespace": "AWS/EC2",
            "AlarmDescription": "CPU usage exceeds 70 percent",
            "StateReasonData": "{\"version\":\"1.0\",\"queryDate\":
\"2014-04-09T18:59:06.419+0000\",\"startDate\":\"2014-04-09T18:44:00.000+0000\",
\"statistic\":\"Average\",\"period\":300,\"recentDatapoints\":[38.958,40.292],
\"threshold\":70.0}",
            "Period": 300,
            "StateValue": "OK",
            "Threshold": 70.0,
            "AlarmName": "myHighCpuAlarm",
            "Dimensions": [
                {
                    "Name": "InstanceId",
                    "Value": "i-0c986c72"
                }
            ],
            "Statistic": "Average",
            "StateReason": "Threshold Crossed: 2 datapoints were not greater than
 the threshold (70.0). The most recent datapoints: [38.958, 40.292].",
            "InsufficientDataActions": [],
            "OKActions": [],
            "ActionsEnabled": false,
```

```
            "MetricName": "CPUUtilization"
        }
    ]
}
```

For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

- For API details, see DescribeAlarmsForMetric in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Checks for a metric alarm in AWS CloudWatch.
    *
    * @param fileName the name of the file containing the JSON configuration for
the custom metric
    * @return a {@link CompletableFuture} that completes when the check for the
metric alarm is complete
    */
   public CompletableFuture<Void> checkForMetricAlarmAsync(String fileName) {
       CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
               return rootNode.toString(); // Return JSON as a string for
further processing
           } catch (IOException e) {
               throw new RuntimeException("Failed to read file", e);
           }
```

```
        });

        return readFileFuture.thenCompose(jsonContent -> {
            try {
                com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                DescribeAlarmsForMetricRequest metricRequest =
DescribeAlarmsForMetricRequest.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .build();

                return checkForAlarmAsync(metricRequest, customMetricName, 10);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error checking metric alarm",
exception);
            }
        });
    }

    // Recursive method to check for the alarm.

    /**
     * Checks for the existence of an alarm asynchronously for the specified
metric.
     *
     * @param metricRequest    the request to describe the alarms for the
specified metric
     * @param customMetricName the name of the custom metric to check for an
alarm
     * @param retries          the number of retries to perform if no alarm is
found
     * @return a {@link CompletableFuture} that completes when an alarm is found
or the maximum number of retries has been reached
```

```
    */
   private static CompletableFuture<Void>
checkForAlarmAsync(DescribeAlarmsForMetricRequest metricRequest, String
customMetricName, int retries) {
       if (retries == 0) {
           return CompletableFuture.completedFuture(null).thenRun(() ->
               logger.info("No Alarm state found for {} after 10 retries.",
customMetricName)
           );
       }

       return
(getAsyncClient().describeAlarmsForMetric(metricRequest).thenCompose(response ->
{
           if (response.hasMetricAlarms()) {
               logger.info("Alarm state found for {}", customMetricName);
               return CompletableFuture.completedFuture(null); // Alarm found,
complete the future
           } else {
               return CompletableFuture.runAsync(() -> {
                   try {
                       Thread.sleep(20000);
                       logger.info(".");
                   } catch (InterruptedException e) {
                       throw new RuntimeException("Interrupted while waiting to
retry", e);
                   }
               }).thenCompose(v -> checkForAlarmAsync(metricRequest,
customMetricName, retries - 1)); // Recursive call
           }
       }));
   }
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DescribeAlarmsForMetric in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DescribeAlarmsForMetric in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeAlarmsForMetric(
    namespaceVal: String,
    metricNameVal: String,
) {
    val request =
        DescribeAlarmsForMetricRequest {
            namespace = namespaceVal
            metricName = metricNameVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAlarmsForMetric(request)
        val alarms = response.metricAlarms
        if (alarms.isNullOrEmpty()) {
            println("No alarms found for $metricNameVal in $namespaceVal.")
        } else {
            for (alarm in alarms) {
                println("Alarm name: ${alarm.alarmName}")
                println("Alarm state: ${alarm.stateValue}")
            }
        }
    }
}
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
```

```
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
 metric_name)
        return alarm_iter
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @example
#   describe_metric_alarms(Aws::CloudWatch::Client.new(region: 'us-east-1'))
def describe_metric_alarms(cloudwatch_client)
  response = cloudwatch_client.describe_alarms
```

```
if response.metric_alarms.count.positive?
  response.metric_alarms.each do |alarm|
    puts '-' * 16
    puts "Name:           #{alarm.alarm_name}"
    puts "State value:    #{alarm.state_value}"
    puts "State reason:   #{alarm.state_reason}"
    puts "Metric:         #{alarm.metric_name}"
    puts "Namespace:      #{alarm.namespace}"
    puts "Statistic:      #{alarm.statistic}"
    puts "Period:         #{alarm.period}"
    puts "Unit:           #{alarm.unit}"
    puts "Eval. periods:  #{alarm.evaluation_periods}"
    puts "Threshold:      #{alarm.threshold}"
    puts "Comp. operator: #{alarm.comparison_operator}"

    if alarm.key?(:ok_actions) && alarm.ok_actions.count.positive?
      puts 'OK actions:'
      alarm.ok_actions.each do |a|
        puts "  #{a}"
      end
    end

    if alarm.key?(:alarm_actions) && alarm.alarm_actions.count.positive?
      puts 'Alarm actions:'
      alarm.alarm_actions.each do |a|
        puts "  #{a}"
      end
    end

    if alarm.key?(:insufficient_data_actions) &&
       alarm.insufficient_data_actions.count.positive?
      puts 'Insufficient data actions:'
      alarm.insufficient_data_actions.each do |a|
        puts "  #{a}"
      end
    end

    puts 'Dimensions:'
    if alarm.key?(:dimensions) && alarm.dimensions.count.positive?
      alarm.dimensions.each do |d|
        puts "  Name: #{d.name}, Value: #{d.value}"
      end
    else
```

```
        puts '  None for this alarm.'
      end
    end
  else
    puts 'No alarms found.'
  end
rescue StandardError => e
  puts "Error getting information about alarms: #{e.message}"
end

# Example usage:
def run_me
  region = ''

  # Print usage information and then stop.
  if ARGV[0] == '--help' || ARGV[0] == '-h'
    puts 'Usage:   ruby cw-ruby-example-show-alarms.rb REGION'
    puts 'Example: ruby cw-ruby-example-show-alarms.rb us-east-1'
    exit 1
  # If no values are specified at the command prompt, use these default values.
  elsif ARGV.count.zero?
    region = 'us-east-1'
  # Otherwise, use the values as specified at the command prompt.
  else
    region = ARGV[0]
  end

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
  puts 'Available alarms:'
  describe_metric_alarms(cloudwatch_client)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_cwt->describealarmsformetric(          " oo_result is
 returned for testing purposes. "
          iv_namespace   = iv_namespace
          iv_metricname  = iv_metric_name
          it_dimensions  = it_dimensions
          iv_statistic   = iv_statistic
          iv_period      = iv_period
          iv_unit        = iv_unit ).
        MESSAGE 'Alarms for metric retrieved.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAnomalyDetectors` with an AWS SDK or CLI

The following code examples show how to use `DescribeAnomalyDetectors`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- For API details, see DescribeAnomalyDetectors in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve a list of anomaly detection models

The following `describe-anomaly-detectors` example displays information about anomaly detector models that are associated with the `AWS/Logs` namespace in the specified account.

```
aws cloudwatch describe-anomaly-detectors \
    --namespace AWS/Logs
```

Output:

```
{
    "AnomalyDetectors": [
        {
            "Namespace": "AWS/Logs",
            "MetricName": "IncomingBytes",
            "Dimensions": [],
            "Stat": "SampleCount",
            "Configuration": {
                "ExcludedTimeRanges": []
            },
            "StateValue": "TRAINED",
            "SingleMetricAnomalyDetector": {
                "AccountId": "123456789012",
                "Namespace": "AWS/Logs",
                "MetricName": "IncomingBytes",
                "Dimensions": [],
                "Stat": "SampleCount"
            }
        },
        {
            "Namespace": "AWS/Logs",
            "MetricName": "IncomingBytes",
            "Dimensions": [
                {
                    "Name": "LogGroupName",
                    "Value": "demo"
                }
            ],
```

```
            "Stat": "Average",
            "Configuration": {
                "ExcludedTimeRanges": []
            },
            "StateValue": "PENDING_TRAINING",
            "SingleMetricAnomalyDetector": {
                "AccountId": "123456789012",
                "Namespace": "AWS/Logs",
                "MetricName": "IncomingBytes",
                "Dimensions": [
                    {
                        "Name": "LogGroupName",
                        "Value": "demo"
                    }
                ],
                "Stat": "Average"
            }
        }
    ]
}
```

For more information, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

- For API details, see DescribeAnomalyDetectors in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Describes the anomaly detectors based on the specified JSON file.
     *
     * @param fileName the name of the JSON file containing the custom metric
namespace and name
```

```
    * @return a {@link CompletableFuture} that completes when the anomaly
detectors have been described
    * @throws RuntimeException if there is a failure during the operation, such
as when reading or parsing the JSON file,
    *                          or when describing the anomaly detectors
    */
   public CompletableFuture<Void> describeAnomalyDetectorsAsync(String fileName)
{
       CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               return new ObjectMapper().readTree(parser);
           } catch (IOException e) {
               throw new RuntimeException("Failed to read or parse the file",
e);
           }
       });

       return readFileFuture.thenCompose(rootNode -> {
           try {
               String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
               String customMetricName =
rootNode.findValue("customMetricName").asText();

               DescribeAnomalyDetectorsRequest detectorsRequest =
DescribeAnomalyDetectorsRequest.builder()
                   .maxResults(10)
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

               return
getAsyncClient().describeAnomalyDetectors(detectorsRequest).thenAccept(response
-> {
                   List<AnomalyDetector> anomalyDetectorList =
response.anomalyDetectors();
                   for (AnomalyDetector detector : anomalyDetectorList) {
                       logger.info("Metric name: {} ",
detector.singleMetricAnomalyDetector().metricName());
                       logger.info("State: {} ", detector.stateValue());
                   }
```

```
                });
            } catch (RuntimeException e) {
                throw new RuntimeException("Failed to describe anomaly
 detectors", e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
                throw new RuntimeException("Error describing anomaly detectors",
 exception);
            }
        });
    }
```

- For API details, see DescribeAnomalyDetectors in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeAnomalyDetectors(
    namespaceVal: String,
    metricNameVal: String,
) {
    val request =
        DescribeAnomalyDetectorsRequest {
            maxResults = 10
            namespace = namespaceVal
            metricName = metricNameVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.describeAnomalyDetectors(request)
        response.anomalyDetectors?.forEach { detector ->
            println("Metric name:
 ${detector.singleMetricAnomalyDetector?.metricName}")
```

```
            println("State: ${detector.stateValue}")
        }
    }
}
```

- For API details, see DescribeAnomalyDetectors in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DisableAlarmActions` with an AWS SDK or CLI

The following code examples show how to use `DisableAlarmActions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Get started with alarms
- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Disable the actions for a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DisableAlarmActions(List<string> alarmNames)
{
```

```
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DisableAlarmActions in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/DisableAlarmActionsRequest.h>
#include <iostream>
```

Disable the alarm actions.

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::DisableAlarmActionsRequest
disableAlarmActionsRequest;
        disableAlarmActionsRequest.AddAlarmNames(alarm_name);

        auto disableAlarmActionsOutcome =
cw.DisableAlarmActions(disableAlarmActionsRequest);
```

```
        if (!disableAlarmActionsOutcome.IsSuccess())
        {
            std::cout << "Failed to disable actions for alarm " << alarm_name <<
                ": " << disableAlarmActionsOutcome.GetError().GetMessage() <<
                std::endl;
        }
        else
        {
            std::cout << "Successfully disabled actions for alarm " <<
                alarm_name << std::endl;
        }
```

- For API details, see DisableAlarmActions in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To disable actions for an alarm

The following `disable-alarm-actions` example disables all actions for the alarm named myalarm.

```
aws cloudwatch disable-alarm-actions \
    --alarm-names myalarm
```

This command produces no output.

This command returns to the prompt if successful.

For more information, see Alarm actions in the *Amazon CloudWatch User Guide*.

- For API details, see DisableAlarmActions in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DisableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DisableAlarmActions {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to disable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarmName = args[0];
```

```
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        disableActions(cw, alarmName);
        cw.close();
    }

    public static void disableActions(CloudWatchClient cw, String alarmName) {
        try {
            DisableAlarmActionsRequest request =
 DisableAlarmActionsRequest.builder()
                    .alarmNames(alarmName)
                    .build();

            cw.disableAlarmActions(request);
            System.out.printf("Successfully disabled actions on alarm %s",
 alarmName);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DisableAlarmActions in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DisableAlarmActions in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DisableAlarmActions in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun disableActions(alarmName: String) {
    val request =
        DisableAlarmActionsRequest {
            alarmNames = listOf(alarmName)
        }
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.disableAlarmActions(request)
        println("Successfully disabled actions on alarm $alarmName")
    }
}
```

- For API details, see DisableAlarmActions in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
 they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
```

```
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- For API details, see DisableAlarmActions in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Disables an alarm in Amazon CloudWatch.
#
# Prerequisites.
#
# - The alarm to disable.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm to disable.
# @return [Boolean] true if the alarm was disabled; otherwise, false.
# @example
#   exit 1 unless alarm_actions_disabled?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket'
#   )
def alarm_actions_disabled?(cloudwatch_client, alarm_name)
  cloudwatch_client.disable_alarm_actions(alarm_names: [alarm_name])
  true
```

```
rescue StandardError => e
  puts "Error disabling alarm actions: #{e.message}"
  false
end

# Example usage:
def run_me
  alarm_name = 'ObjectsInBucket'
  alarm_description = 'Objects exist in this bucket for more than 1 day.'
  metric_name = 'NumberOfObjects'
  # Notify this Amazon Simple Notification Service (Amazon SNS) topic when
  # the alarm transitions to the ALARM state.
  alarm_actions = ['arn:aws:sns:us-
east-1:111111111111:Default_CloudWatch_Alarms_Topic']
  namespace = 'AWS/S3'
  statistic = 'Average'
  dimensions = [
    {
      name: "BucketName",
      value: "amzn-s3-demo-bucket"
    },
    {
      name: 'StorageType',
      value: 'AllStorageTypes'
    }
  ]
  period = 86_400 # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
  unit = 'Count'
  evaluation_periods = 1 # More than one day.
  threshold = 1 # One object.
  comparison_operator = 'GreaterThanThreshold' # More than one object.
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = 'us-east-1'

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  if alarm_created_or_updated?(
    cloudwatch_client,
    alarm_name,
    alarm_description,
    metric_name,
    alarm_actions,
    namespace,
    statistic,
```

```
    dimensions,
    period,
    unit,
    evaluation_periods,
    threshold,
    comparison_operator
  )
    puts "Alarm '#{alarm_name}' created or updated."
  else
    puts "Could not create or update alarm '#{alarm_name}'."
  end

  if alarm_actions_disabled?(cloudwatch_client, alarm_name)
    puts "Alarm '#{alarm_name}' disabled."
  else
    puts "Could not disable alarm '#{alarm_name}'."
  end
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see DisableAlarmActions in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    "Disables actions on the specified alarm. "
    TRY.
        lo_cwt->disablealarmactions(
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarm actions disabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
```

```
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see DisableAlarmActions in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `EnableAlarmActions` with an AWS SDK or CLI

The following code examples show how to use `EnableAlarmActions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
```

```
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see EnableAlarmActions in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/EnableAlarmActionsRequest.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Enable the alarm actions.

```
    Aws::CloudWatch::CloudWatchClient cw;
    Aws::CloudWatch::Model::PutMetricAlarmRequest request;
    request.SetAlarmName(alarm_name);
    request.SetComparisonOperator(
        Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
    request.SetEvaluationPeriods(1);
    request.SetMetricName("CPUUtilization");
    request.SetNamespace("AWS/EC2");
    request.SetPeriod(60);
```

```
    request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
    request.SetThreshold(70.0);
    request.SetActionsEnabled(false);
    request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
    request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);
    request.AddAlarmActions(actionArn);

    Aws::CloudWatch::Model::Dimension dimension;
    dimension.SetName("InstanceId");
    dimension.SetValue(instanceId);
    request.AddDimensions(dimension);

    auto outcome = cw.PutMetricAlarm(request);
    if (!outcome.IsSuccess())
    {
        std::cout << "Failed to create CloudWatch alarm:" <<
            outcome.GetError().GetMessage() << std::endl;
        return;
    }

    Aws::CloudWatch::Model::EnableAlarmActionsRequest enable_request;
    enable_request.AddAlarmNames(alarm_name);

    auto enable_outcome = cw.EnableAlarmActions(enable_request);
    if (!enable_outcome.IsSuccess())
    {
        std::cout << "Failed to enable alarm actions:" <<
            enable_outcome.GetError().GetMessage() << std::endl;
        return;
    }

    std::cout << "Successfully created alarm " << alarm_name <<
        " and enabled actions on it." << std::endl;
```

- For API details, see EnableAlarmActions in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To enable all actions for an alarm

The following `enable-alarm-actions` example enables all actions for the alarm named myalarm.

```
aws cloudwatch enable-alarm-actions \
    --alarm-names myalarm
```

This command produces no output.

This command returns to the prompt if successful.

For more information, see Alarm actions in the *Amazon CloudWatch User Guide*.

- For API details, see EnableAlarmActions in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.EnableAlarmActionsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnableAlarmActions {
    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:
                  <alarmName>

                Where:
                  alarmName - An alarm name to enable (for example, MyAlarm).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String alarm = args[0];
        Region region = Region.US_EAST_1;
        CloudWatchClient cw = CloudWatchClient.builder()
                .region(region)
                .build();

        enableActions(cw, alarm);
        cw.close();
    }

    public static void enableActions(CloudWatchClient cw, String alarm) {
        try {
            EnableAlarmActionsRequest request =
 EnableAlarmActionsRequest.builder()
                    .alarmNames(alarm)
                    .build();

            cw.enableAlarmActions(request);
            System.out.printf("Successfully enabled actions on alarm %s", alarm);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see EnableAlarmActions in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see EnableAlarmActions in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see EnableAlarmActions in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun enableActions(alarm: String) {
    val request =
        EnableAlarmActionsRequest {
            alarmNames = listOf(alarm)
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.enableAlarmActions(request)
        println("Successfully enabled actions on alarm $alarm")
    }
}
```

- For API details, see EnableAlarmActions in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def enable_alarm_actions(self, alarm_name, enable):
        """
        Enables or disables actions on the specified alarm. Alarm actions can be
        used to send notifications or automate responses when an alarm enters a
        particular state.

        :param alarm_name: The name of the alarm.
        :param enable: When True, actions are enabled for the alarm. Otherwise,
 they
                       disabled.
        """
        try:
            alarm = self.cloudwatch_resource.Alarm(alarm_name)
            if enable:
                alarm.enable_actions()
            else:
                alarm.disable_actions()
            logger.info(
                "%s actions for alarm %s.",
                "Enabled" if enable else "Disabled",
```

```
                alarm_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't %s actions alarm %s.",
                "enable" if enable else "disable",
                alarm_name,
            )
            raise
```

- For API details, see EnableAlarmActions in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    "Enable actions on the specified alarm."
    TRY.
        lo_cwt->enablealarmactions(
          it_alarmnames = it_alarm_names ).
        MESSAGE 'Alarm actions enabled.' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- For API details, see EnableAlarmActions in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAlarmMuteRule` with an AWS SDK

The following code examples show how to use `GetAlarmMuteRule`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get the full configuration of an alarm mute rule, including its schedule,
the
   /// alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.
   /// </summary>
   /// <param name="name">The name of the mute rule.</param>
   /// <returns>The mute rule.</returns>
   public async Task<GetAlarmMuteRuleResponse> GetAlarmMuteRule(string name)
   {
       var response = await _amazonCloudWatch.GetAlarmMuteRuleAsync(
           new GetAlarmMuteRuleRequest
           {
               AlarmMuteRuleName = name
           });
```

```
        _logger.LogInformation($"Mute rule {response.Name} is
{response.Status}.");
        return response;
    }
```

- For API details, see GetAlarmMuteRule in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/AlarmMuteRuleStatus.h>
#include <aws/monitoring/model/GetAlarmMuteRuleRequest.h>
#include <iostream>
```

Get the alarm mute rule.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::GetAlarmMuteRuleRequest request;
        request.SetAlarmMuteRuleName(mute_rule_name);

        auto outcome = cw.GetAlarmMuteRule(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to get alarm mute rule: "
                      << outcome.GetError().GetMessage() << std::endl;
```

```
        } else {
            const auto &result = outcome.GetResult();
            std::cout << "Mute rule " << result.GetName() << " is "
                      << Aws::CloudWatch::Model::AlarmMuteRuleStatusMapper::
                             GetNameForAlarmMuteRuleStatus(result.GetStatus())
                      << "." << std::endl;
            std::cout << "  ARN: " << result.GetAlarmMuteRuleArn() << std::endl;
            std::cout << "  schedule: " <<
result.GetRule().GetSchedule().GetExpression()
                      << " for " << result.GetRule().GetSchedule().GetDuration()
                      << std::endl;

            const auto &alarm_names = result.GetMuteTargets().GetAlarmNames();
            if (!alarm_names.empty()) {
                std::cout << "  muted alarms:";
                for (const auto &alarm_name : alarm_names) {
                    std::cout << " " << alarm_name;
                }
                std::cout << std::endl;
            }
        }
```

- For API details, see GetAlarmMuteRule in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Gets the full configuration of an alarm mute rule, including its schedule,
the
     * alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.
     *
```

```
     * @param cw   the CloudWatch client
     * @param name the name of the mute rule
     * @return the mute rule
     */
    public static GetAlarmMuteRuleResponse getAlarmMuteRule(CloudWatchClient cw,
String name) {
        try {
            GetAlarmMuteRuleResponse response =
cw.getAlarmMuteRule(GetAlarmMuteRuleRequest.builder()
                    .alarmMuteRuleName(name)
                    .build());

            System.out.printf("Mute rule %s is %s.%n", response.name(),
response.statusAsString());
            return response;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return null;
        }
    }
```

- For API details, see GetAlarmMuteRule in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { GetAlarmMuteRuleCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Get the full configuration of an alarm mute rule, including its schedule, the
 alarms
// it targets, and whether it is currently SCHEDULED, ACTIVE, or EXPIRED.
```

```
const run = async () => {
  const command = new GetAlarmMuteRuleCommand({
    AlarmMuteRuleName: process.env.CLOUDWATCH_MUTE_RULE_NAME, // Set
 CLOUDWATCH_MUTE_RULE_NAME to the name of an existing mute rule.
  });

  try {
    const response = await client.send(command);
    console.log(`Mute rule ${response.Name} is ${response.Status}.`);
    return response;
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see GetAlarmMuteRule in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAlarmMuteRule(muteRuleName: String): GetAlarmMuteRuleResponse {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response =
            cwClient.getAlarmMuteRule(
                GetAlarmMuteRuleRequest {
                    alarmMuteRuleName = muteRuleName
                },
            )

        println("Mute rule ${response.name} is ${response.status?.value}")
        println("  ARN: ${response.alarmMuteRuleArn}")
```

```
        println("  schedule: ${response.rule?.schedule?.expression} for
 ${response.rule?.schedule?.duration}")
        response.muteTargets?.alarmNames?.let { println("  muted alarms:
 ${it.joinToString(", ")}") }
        return response
    }
}
```

- For API details, see GetAlarmMuteRule in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))
```

```
    def get_alarm_mute_rule(self, name):
        """
        Gets the full configuration of an alarm mute rule, including its
schedule, the
        alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.

        :param name: The name of the mute rule.
        :return: The mute rule.
        """
        try:
            response = self.cloudwatch_client.get_alarm_mute_rule(
                AlarmMuteRuleName=name
            )
        except ClientError:
            logger.exception("Couldn't get alarm mute rule %s.", name)
            raise
        else:
            logger.info("Got alarm mute rule %s.", name)
            return response
```

- For API details, see GetAlarmMuteRule in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Gets the full configuration of an alarm mute rule, including its schedule, the
 alarms
# it targets, and whether it is currently SCHEDULED, ACTIVE, or EXPIRED.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
```

```
# @param name [String] The name of the mute rule.
# @return [Aws::CloudWatch::Types::GetAlarmMuteRuleOutput, nil] The mute rule, or
 nil on
#   error.
def alarm_mute_rule(cloudwatch_client, name)
  cloudwatch_client.get_alarm_mute_rule(alarm_mute_rule_name: name)
rescue StandardError => e
  puts "Error getting alarm mute rule: #{e.message}"
  nil
end
```

- For API details, see GetAlarmMuteRule in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetDashboard` with an AWS SDK or CLI

The following code examples show how to use `GetDashboard`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Creating a monitoring dashboard with function name as a variable
- Using property variables in monitoring dashboards to monitor multiple serverless functions

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
```

```
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }
```

- For API details, see GetDashboard in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve information about a Dashboard

The following `get-dashboard` example displays information about the dashboard named `Dashboard-A` in the specified account.

```
aws cloudwatch get-dashboard \
    --dashboard-name Dashboard-A
```

Output:

```
{
    "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/Dashboard-A",
    "DashboardBody": "{\"widgets\":[{\"type\":\"metric\",\"x\":0,\"y\":0,\"width
\":6,\"height\":6,\"properties\":{\"view\":\"timeSeries\",\"stacked\":false,
\"metrics\":[[\"AWS/EC2\",\"NetworkIn\",\"InstanceId\",\"i-0131f062232ade043\"],
[\".\",\"NetworkOut\",\".\",\".\"]],\"region\":\"us-west-2\"}}]}",
    "DashboardName": "Dashboard-A"
}
```

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see GetDashboard in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns the arn the body of the specified dashboard.

```
Get-CWDashboard -DashboardName Dashboard1
```

Output:

```
DashboardArn                                              DashboardBody
------------                                              -------------
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- For API details, see GetDashboard in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns the arn the body of the specified dashboard.

```
Get-CWDashboard -DashboardName Dashboard1
```

Output:

```
DashboardArn                                              DashboardBody
------------                                              -------------
arn:aws:cloudwatch::123456789012:dashboard/Dashboard1 {...
```

- For API details, see GetDashboard in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricData` with an AWS SDK or CLI

The following code examples show how to use `GetMetricData`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Run CPU stress tests on virtual machine instances using fault injection

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData,
 bool useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
```

```
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTime = startTimeUtc,
                EndTime = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- For API details, see GetMetricData in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

Example 1: To get the Average Total IOPS for the specified EC2 using math expression

The following `get-metric-data` example retrieves CloudWatch metric values for the EC2 instance with InstanceID `i-abcdef` using metric math exprssion that combines `EBSReadOps` and `EBSWriteOps` metrics.

```
aws cloudwatch get-metric-data \
```

```
    --metric-data-queries file://file.json \
    --start-time 2024-09-29T22:10:00Z \
    --end-time 2024-09-29T22:15:00Z
```

Contents of `file.json`:

```
[
    {
        "Id": "m3",
        "Expression": "(m1+m2)/300",
        "Label": "Avg Total IOPS"
    },
    {
        "Id": "m1",
        "MetricStat": {
            "Metric": {
                "Namespace": "AWS/EC2",
                "MetricName": "EBSReadOps",
                "Dimensions": [
                    {
                        "Name": "InstanceId",
                        "Value": "i-abcdef"
                    }
                ]
            },
            "Period": 300,
            "Stat": "Sum",
            "Unit": "Count"
        },
        "ReturnData": false
    },
    {
        "Id": "m2",
        "MetricStat": {
            "Metric": {
                "Namespace": "AWS/EC2",
                "MetricName": "EBSWriteOps",
                "Dimensions": [
                    {
                        "Name": "InstanceId",
                        "Value": "i-abcdef"
                    }
                ]
```

```
            },
            "Period": 300,
            "Stat": "Sum",
            "Unit": "Count"
        },
        "ReturnData": false
    }
]
```

Output:

```
{
    "MetricDataResults": [
        {
            "Id": "m3",
            "Label": "Avg Total IOPS",
            "Timestamps": [
                "2024-09-29T22:10:00+00:00"
            ],
            "Values": [
                96.85
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

Example 2: To monitor the estimated AWS charges using CloudWatch billing metrics

The following `get-metric-data` example retrieves `EstimatedCharges` CloudWatch metric from AWS/Billing namespace.

```
aws cloudwatch get-metric-data \
    --metric-data-queries '[{"Id":"m1","MetricStat":{"Metric":
{"Namespace":"AWS/Billing","MetricName":"EstimatedCharges","Dimensions":
[{"Name":"Currency","Value":"USD"}]},"Period":21600,"Stat":"Maximum"}}]' \
    --start-time 2024-09-26T12:00:00Z \
    --end-time 2024-09-26T18:00:00Z \
    --region us-east-1
```

Output:

```
{
    "MetricDataResults": [
        {
            "Id": "m1",
            "Label": "EstimatedCharges",
            "Timestamps": [
                "2024-09-26T12:00:00+00:00"
            ],
            "Values": [
                542.38
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

For more information, see Using math expressions with CloudWatch metrics in the *Amazon CloudWatch User Guide*.

- For API details, see GetMetricData in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Retrieves custom metric data from the AWS CloudWatch service.
     *
     * @param fileName the name of the file containing the custom metric
information
     * @return a {@link CompletableFuture} that completes when the metric data
has been retrieved
     */
```

```
   public CompletableFuture<Void> getCustomMetricDataAsync(String fileName) {
       CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               // Read values from the JSON file.
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
               return rootNode.toString(); // Return JSON as a string for
further processing
           } catch (IOException e) {
               throw new RuntimeException("Failed to read file", e);
           }
       });

       return readFileFuture.thenCompose(jsonContent -> {
           try {
               // Parse the JSON string to extract relevant values.
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
               String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
               String customMetricName =
rootNode.findValue("customMetricName").asText();

               // Set the current time and date range for metric query.
               Instant nowDate = Instant.now();
               long hours = 1;
               long minutes = 30;
               Instant endTime = nowDate.plus(hours,
ChronoUnit.HOURS).plus(minutes, ChronoUnit.MINUTES);

               Metric met = Metric.builder()
                   .metricName(customMetricName)
                   .namespace(customMetricNamespace)
                   .build();

               MetricStat metStat = MetricStat.builder()
                   .stat("Maximum")
                   .period(60)  // Assuming period in seconds
                   .metric(met)
                   .build();
```

```
                MetricDataQuery dataQuery = MetricDataQuery.builder()
                    .metricStat(metStat)
                    .id("foo2")
                    .returnData(true)
                    .build();

                List<MetricDataQuery> dq = new ArrayList<>();
                dq.add(dataQuery);

                GetMetricDataRequest getMetricDataRequest =
GetMetricDataRequest.builder()
                    .maxDatapoints(10)
                    .scanBy(ScanBy.TIMESTAMP_DESCENDING)
                    .startTime(nowDate)
                    .endTime(endTime)
                    .metricDataQueries(dq)
                    .build();

                // Call the async method for CloudWatch data retrieval.
                return getAsyncClient().getMetricData(getMetricDataRequest);

            } catch (IOException e) {
                throw new RuntimeException("Failed to parse JSON content", e);
            }
        }).thenAccept(response -> {
            List<MetricDataResult> data = response.metricDataResults();
            for (MetricDataResult item : data) {
                logger.info("The label is: {}", item.label());
                logger.info("The status code is: {}",
item.statusCode().toString());
            }
        }).exceptionally(exception -> {
            throw new RuntimeException("Failed to get metric data", exception);
        });
    }
```

- For API details, see GetMetricData in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getMetData() {
    val start =
        aws.smithy.kotlin.runtime.time.Instant
            .fromIso8601("2019-10-23T10:12:35Z")
    val endDate =
        aws.smithy.kotlin.runtime.time.Instant
            .now()
    val met =
        Metric {
            metricName = "DiskReadBytes"
            namespace = "AWS/EC2"
        }

    val metStat =
        MetricStat {
            stat = "Minimum"
            period = 60
            metric = met
        }

    val dataQUery =
        MetricDataQuery {
            metricStat = metStat
            id = "foo2"
            returnData = true
        }

    val dq = mutableListOf<MetricDataQuery>()
    dq.add(dataQUery)

    val request =
        GetMetricDataRequest {
```

```
            maxDatapoints = 100
            startTime = start
            endTime = endDate
            metricDataQueries = dq
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricData(request)
        response.metricDataResults?.forEach { item ->
            println("The label is ${item.label}")
            println("The status code is ${item.statusCode}")
        }
    }
}
```

- For API details, see GetMetricData in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricStatistics` with an AWS SDK or CLI

The following code examples show how to use `GetMetricStatistics`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string
metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions,
int days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTime = DateTime.UtcNow.AddDays(-days),
                EndTime = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }
```

- For API details, see [GetMetricStatistics](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

Example 1: To get the CPU utilization per EC2 instance

The following `get-metric-statistics` example gets the CPU utilization for an EC2 instance with the ID i-abcdef.

```
aws cloudwatch get-metric-statistics \
    --metric-name CPUUtilization \
    --start-time 2014-04-08T23:18:00Z \
    --end-time 2014-04-09T23:18:00Z \
    --period 3600 \
    --namespace AWS/EC2 \
    --statistics Maximum \
    --dimensions Name=InstanceId,Value=i-abcdef
```

Output:

```
{
    "Datapoints": [
        {
            "Timestamp": "2014-04-09T11:18:00Z",
            "Maximum": 44.79,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T20:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T19:18:00Z",
            "Maximum": 50.85,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T09:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T03:18:00Z",
            "Maximum": 76.84,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T21:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
```

```
        {
            "Timestamp": "2014-04-09T14:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T08:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T16:18:00Z",
            "Maximum": 45.55,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T06:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T13:18:00Z",
            "Maximum": 45.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T05:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T18:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T17:18:00Z",
            "Maximum": 52.08,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T07:18:00Z",
            "Maximum": 47.92,
            "Unit": "Percent"
```

```
        },
        {
            "Timestamp": "2014-04-09T02:18:00Z",
            "Maximum": 51.23,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T12:18:00Z",
            "Maximum": 47.67,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-08T23:18:00Z",
            "Maximum": 46.88,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T10:18:00Z",
            "Maximum": 51.91,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T04:18:00Z",
            "Maximum": 47.13,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T15:18:00Z",
            "Maximum": 48.96,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T00:18:00Z",
            "Maximum": 48.16,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2014-04-09T01:18:00Z",
            "Maximum": 49.18,
            "Unit": "Percent"
        }
    ],
    "Label": "CPUUtilization"
```

```
}
```

For more information, see Get statistics for a metric (GetMetricStatistics) in the *Amazon CloudWatch User Guide*.

Example 2: To get statistics for a metric with multiple dimensions

The following `get-metric-statistics` example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space. If a single metric includes multiple dimensions, you must specify a value for every defined dimension.

```
aws cloudwatch get-metric-statistics \
    --metric-name Buffers \
    --namespace MyNameSpace \
    --dimensions Name=InstanceID,Value=i-abcdef Name=InstanceType,Value=m1.small
 \
    --start-time 2016-10-15T04:00:00Z \
    --end-time 2016-10-19T07:00:00Z \
    --statistics Average \
    --period 60
```

Output:

```
{
    "Label": "Buffers",
    "Datapoints": [
        {
            "Timestamp": "2016-10-15T04:00:00Z",
            "Average": 231434333.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2016-10-15T04:01:00Z",
            "Average": 231434333.0,
            "Unit": "Bytes"
        }
    ]
}
```

For more information, see Get statistics for a metric (GetMetricStatistics) in the *Amazon CloudWatch User Guide*.

- For API details, see GetMetricStatistics in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Retrieves and displays metric statistics for the specified parameters.
    *
    * @param nameSpace    the namespace for the metric
    * @param metVal       the name of the metric
    * @param metricOption the statistic to retrieve for the metric (e.g.,
"Maximum", "Average")
    * @param date         the date for which to retrieve the metric statistics,
in the format "yyyy-MM-dd'T'HH:mm:ss'Z'"
    * @param myDimension  the dimension(s) to filter the metric statistics by
    * @return a {@link CompletableFuture} that completes when the metric
statistics have been retrieved and displayed
    */
   public CompletableFuture<GetMetricStatisticsResponse>
getAndDisplayMetricStatisticsAsync(String nameSpace, String metVal,

           String metricOption, String date, Dimension myDimension) {

       Instant start = Instant.parse(date);
       Instant endDate = Instant.now();

       // Building the request for metric statistics.
       GetMetricStatisticsRequest statisticsRequest =
GetMetricStatisticsRequest.builder()
           .endTime(endDate)
           .startTime(start)
           .dimensions(myDimension)
           .metricName(metVal)
           .namespace(nameSpace)
           .period(86400) // 1 day period
```

```
            .statistics(Statistic.fromValue(metricOption))
            .build();

        return getAsyncClient().getMetricStatistics(statisticsRequest)
            .whenComplete((response, exception) -> {
                if (response != null) {
                    List<Datapoint> data = response.datapoints();
                    if (!data.isEmpty()) {
                        for (Datapoint datapoint : data) {
                            logger.info("Timestamp: {} Maximum value: {}",
datapoint.timestamp(), datapoint.maximum());
                        }
                    } else {
                        logger.info("The returned data list is empty");
                    }
                } else {
                    logger.info("Failed to get metric statistics: {} ",
exception.getMessage());
                }
            })
            .exceptionally(exception -> {
                throw new RuntimeException("Error while getting metric
statistics: " + exception.getMessage(), exception);
            });
    }
```

- For API details, see GetMetricStatistics in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAndDisplayMetricStatistics(
    nameSpaceVal: String,
```

```
    metVal: String,
    metricOption: String,
    date: String,
    myDimension: Dimension,
) {
    val start = Instant.parse(date)
    val endDate = Instant.now()
    val statisticsRequest =
        GetMetricStatisticsRequest {
            endTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(endDate)
            startTime =
                aws.smithy.kotlin.runtime.time
                    .Instant(start)
            dimensions = listOf(myDimension)
            metricName = metVal
            namespace = nameSpaceVal
            period = 86400
            statistics = listOf(Statistic.fromValue(metricOption))
        }

    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricStatistics(statisticsRequest)
        val data = response.datapoints
        if (data != null) {
            if (data.isNotEmpty()) {
                for (datapoint in data) {
                    println("Timestamp: ${datapoint.timestamp} Maximum value:
 ${datapoint.maximum}")
                }
            } else {
                println("The returned data list is empty")
            }
        }
    }
}
```

- For API details, see GetMetricStatistics in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def get_metric_statistics(self, namespace, name, start, end, period,
 stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
 grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
 period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
 must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
 value
                           or maximum value.
        :return: The retrieved statistics for the metric.
```

```
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats
```

- For API details, see GetMetricStatistics in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   TRY.
       oo_result = lo_cwt->getmetricstatistics(              " oo_result is
returned for testing purposes. "
         iv_namespace   = iv_namespace
         iv_metricname  = iv_metric_name
         iv_starttime   = iv_start_time
         iv_endtime     = iv_end_time
         iv_period      = iv_period
         it_statistics  = it_statistics ).
```

```
        MESSAGE 'Metric statistics retrieved.' TYPE 'I'.
      CATCH /aws1/cx_cwtinvparamvalueex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
    ENDTRY.
```

- For API details, see GetMetricStatistics in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetMetricWidgetImage` with an AWS SDK or CLI

The following code examples show how to use `GetMetricWidgetImage`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string
metricNamespace, string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
```

```
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString =
JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- For API details, see GetMetricWidgetImage in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve a snapshot graph of CPUUtilization

The following `get-metric-widget-image` example retrieves snapshot graph for the metric `CPUUtilization` of the EC2 instance with the ID `i-abcde` and saves the retrieved image as a file named "image.png" on your local machine.

```
aws cloudwatch get-metric-widget-image \
    --metric-widget '{"metrics":[["AWS/EC2","CPUUtilization","InstanceId","i-abcde"]]}' \
    --output-format png \
    --output text | base64 --decode > image.png
```

This command produces no output.

For more information, see Graphing metrics in the *Amazon CloudWatch User Guide*.

- For API details, see GetMetricWidgetImage in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Retrieves and saves a custom metric image to a file.
     *
     * @param fileName the name of the file to save the metric image to
     * @return a {@link CompletableFuture} that completes when the image has been saved to the file
     */
    public CompletableFuture<Void> downloadAndSaveMetricImageAsync(String fileName) {
```

```
        logger.info("Getting Image data for custom metric.");
        String myJSON = """
              {
                  "title": "Example Metric Graph",
                  "view": "timeSeries",
                  "stacked ": false,
                  "period": 10,
                  "width": 1400,
                  "height": 600,
                  "metrics": [
                      [
                      "AWS/Billing",
                      "EstimatedCharges",
                      "Currency",
                      "USD"
                     ]
                  ]
              }
             """;

        GetMetricWidgetImageRequest imageRequest =
GetMetricWidgetImageRequest.builder()
            .metricWidget(myJSON)
            .build();

        return getAsyncClient().getMetricWidgetImage(imageRequest)
            .thenCompose(response -> {
                SdkBytes sdkBytes = response.metricWidgetImage();
                byte[] bytes = sdkBytes.asByteArray();
                return CompletableFuture.runAsync(() -> {
                    try {
                        File outputFile = new File(fileName);
                        try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                            outputStream.write(bytes);
                        }
                    } catch (IOException e) {
                        throw new RuntimeException("Failed to write image to
file", e);
                    }
                });
            })
            .whenComplete((result, exception) -> {
                if (exception != null) {
```

```
                    throw new RuntimeException("Error getting and saving metric
 image", exception);
                } else {
                    logger.info("Image data saved successfully to {}", fileName);
                }
            });
    }
```

- For API details, see GetMetricWidgetImage in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAndOpenMetricImage(fileName: String) {
    println("Getting image data for a custom metric.")
    val myJSON = """{
        "title": "Example Metric Graph",
        "view": "timeSeries",
        "stacked ": false,
        "period": 10,
        "width": 1400,
        "height": 600,
        "metrics": [
            [
            "AWS/Billing",
            "EstimatedCharges",
            "Currency",
            "USD"
            ]
        ]
        }"""

    val request =
        GetMetricWidgetImageRequest {
```

```
            metricWidget = myJSON
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getMetricWidgetImage(request)
        val bytes = response.metricWidgetImage
        if (bytes != null) {
            File(fileName).writeBytes(bytes)
            println("You have successfully written data to $fileName.")
        }
    }
}
```

- For API details, see GetMetricWidgetImage in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetOTelEnrichment` with an AWS SDK

The following code examples show how to use `GetOTelEnrichment`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get the current OTel enrichment status for the account.
    /// </summary>
    /// <returns>The status, either Running or Stopped.</returns>
    public async Task<OTelEnrichmentStatus> GetOTelEnrichmentStatus()
    {
        var response = await _amazonCloudWatch.GetOTelEnrichmentAsync(
            new GetOTelEnrichmentRequest());

        _logger.LogInformation($"OTel enrichment status is {response.Status}.");
        return response.Status;
    }
```

- For API details, see GetOTelEnrichment in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/GetOTelEnrichmentRequest.h>
#include <aws/monitoring/model/OTelEnrichmentStatus.h>
#include <iostream>
```

Get the OpenTelemetry enrichment status.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);
```

```
        Aws::CloudWatch::Model::GetOTelEnrichmentRequest request;

        auto outcome = cw.GetOTelEnrichment(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to get OTel enrichment status: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            auto status = outcome.GetResult().GetStatus();
            std::cout << "OTel enrichment status is "
                      << Aws::CloudWatch::Model::OTelEnrichmentStatusMapper::
                             GetNameForOTelEnrichmentStatus(status)
                      << "." << std::endl;

            if (status == Aws::CloudWatch::Model::OTelEnrichmentStatus::Running)
{
                std::cout << "Vended metrics are queryable with PromQL." <<
std::endl;
            } else {
                std::cout << "Start enrichment to enrich vended metrics with
resource "
                             "ARN and tag labels."
                          << std::endl;
            }
        }
```

- For API details, see GetOTelEnrichment in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Gets the current OTel enrichment status for the account.
```

```
     *
     * @param cw the CloudWatch client
     * @return the status, either {@code Running} or {@code Stopped}
     */
    public static String getOTelEnrichmentStatus(CloudWatchClient cw) {
        try {
            String status =
 cw.getOTelEnrichment(GetOTelEnrichmentRequest.builder().build())
                    .statusAsString();
            System.out.printf("OTel enrichment status is %s.%n", status);
            return status;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return null;
        }
    }
```

- For API details, see GetOTelEnrichment in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { GetOTelEnrichmentCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Get the current OTel enrichment status for the account. Status is either
// "Running" or "Stopped".
const run = async () => {
  const command = new GetOTelEnrichmentCommand({});

  try {
    const response = await client.send(command);
```

```
    console.log(`OTel enrichment status is ${response.Status}.`);
    return response;
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see GetOTelEnrichment in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getOTelEnrichmentStatus(): OTelEnrichmentStatus? {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        val response = cwClient.getOTelEnrichment(GetOTelEnrichmentRequest {})
        val status = response.status
        when (status) {
            is OTelEnrichmentStatus.Running ->
                println("OTel enrichment is running. Vended metrics are queryable
 with PromQL")
            is OTelEnrichmentStatus.Stopped ->
                println("OTel enrichment is stopped. Start it to enrich vended
 metrics")
            else -> println("OTel enrichment status is ${status?.value}")
        }
        return status
    }
}
```

- For API details, see GetOTelEnrichment in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def get_otel_enrichment_status(self):
        """
        Gets the current OTel enrichment status for the account.

        :return: The status, either 'Running' or 'Stopped'.
        """
        try:
            response = self.cloudwatch_client.get_o_tel_enrichment()
        except ClientError:
            logger.exception("Couldn't get the OTel enrichment status.")
```

```
            raise
        else:
            status = response["Status"]
            logger.info("OTel enrichment status is %s.", status)
            return status
```

- For API details, see GetOTelEnrichment in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Gets the current OTel enrichment status for the account.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [String, nil] 'Running' or 'Stopped', or nil if the status could not be
 read.
def otel_enrichment_status(cloudwatch_client)
  cloudwatch_client.get_o_tel_enrichment.status
rescue StandardError => e
  puts "Error getting OTel enrichment status: #{e.message}"
  nil
end
```

- For API details, see GetOTelEnrichment in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListAlarmMuteRules` with an AWS SDK

The following code examples show how to use `ListAlarmMuteRules`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// List the alarm mute rules in the account, optionally filtered to the
rules that
   /// target one alarm.
   /// </summary>
   /// <param name="alarmName">When specified, only rules that target this alarm
are
   /// returned.</param>
   /// <returns>The list of mute rule summaries.</returns>
   public async Task<List<AlarmMuteRuleSummary>> ListAlarmMuteRules(string?
alarmName = null)
   {
       var results = new List<AlarmMuteRuleSummary>();
       string? nextToken = null;

       do
       {
           var response = await _amazonCloudWatch.ListAlarmMuteRulesAsync(
               new ListAlarmMuteRulesRequest
               {
                   AlarmName = alarmName,
```

```
                    NextToken = nextToken
                });

            if (response.AlarmMuteRuleSummaries != null)
            {
                results.AddRange(response.AlarmMuteRuleSummaries);
            }

            nextToken = response.NextToken;
        } while (!string.IsNullOrEmpty(nextToken));

        _logger.LogInformation($"Got {results.Count} alarm mute rules.");
        return results;
    }
```

- For API details, see ListAlarmMuteRules in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/AlarmMuteRuleStatus.h>
#include <aws/monitoring/model/ListAlarmMuteRulesRequest.h>
#include <iostream>
```

List the alarm mute rules.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
```

```
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::ListAlarmMuteRulesRequest request;
        if (argc == 2) {
            request.SetAlarmName(argv[1]);
        }

        bool done = false;
        while (!done) {
            auto outcome = cw.ListAlarmMuteRules(request);
            if (!outcome.IsSuccess()) {
                std::cerr << "Failed to list alarm mute rules: "
                          << outcome.GetError().GetMessage() << std::endl;
                break;
            }

            for (const auto &summary :
outcome.GetResult().GetAlarmMuteRuleSummaries()) {
                std::cout << summary.GetAlarmMuteRuleArn() << " ("
                          << Aws::CloudWatch::Model::AlarmMuteRuleStatusMapper::

GetNameForAlarmMuteRuleStatus(summary.GetStatus())
                          << ")" << std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- For API details, see ListAlarmMuteRules in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Lists the alarm mute rules in the account, optionally filtered to the
rules that
    * target one alarm.
    *
    * @param cw        the CloudWatch client
    * @param alarmName when non-null, only rules that target this alarm are
returned
    * @return the list of mute rule summaries
    */
   public static List<AlarmMuteRuleSummary> listAlarmMuteRules(CloudWatchClient
cw, String alarmName) {
       List<AlarmMuteRuleSummary> summaries = new ArrayList<>();
       try {
           String nextToken = null;
           do {
               ListAlarmMuteRulesRequest request =
ListAlarmMuteRulesRequest.builder()
                       .alarmName(alarmName)
                       .nextToken(nextToken)
                       .build();

               ListAlarmMuteRulesResponse response =
cw.listAlarmMuteRules(request);
               summaries.addAll(response.alarmMuteRuleSummaries());
               nextToken = response.nextToken();
           } while (nextToken != null && !nextToken.isEmpty());

           for (AlarmMuteRuleSummary summary : summaries) {
               System.out.printf("%s (%s)%n", summary.alarmMuteRuleArn(),
summary.statusAsString());
           }
           return summaries;

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
           return summaries;
       }
   }
```

- For API details, see ListAlarmMuteRules in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { ListAlarmMuteRulesCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// List the alarm mute rules in the account. Filter by the alarm they target, by
// status, or both.
const run = async () => {
  const summaries = [];
  let nextToken;

  try {
    do {
      const command = new ListAlarmMuteRulesCommand({
        AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set
 CLOUDWATCH_ALARM_NAME to filter to rules targeting one alarm.
        Statuses: ["SCHEDULED", "ACTIVE"], // Valid values: SCHEDULED, ACTIVE,
 EXPIRED.
        NextToken: nextToken,
      });
      const response = await client.send(command);
      summaries.push(...(response.AlarmMuteRuleSummaries ?? []));
      nextToken = response.NextToken;
    } while (nextToken);

    for (const summary of summaries) {
      console.log(`${summary.AlarmMuteRuleArn} (${summary.Status})`);
    }
    return summaries;
  } catch (err) {
    console.error(err);
  }
```

```
};

export default run();
```

- For API details, see ListAlarmMuteRules in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listAlarmMuteRules(alarmNameVal: String? = null):
 List<AlarmMuteRuleSummary> {
    val summaries = mutableListOf<AlarmMuteRuleSummary>()

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        var token: String? = null
        do {
            val response =
                cwClient.listAlarmMuteRules(
                    ListAlarmMuteRulesRequest {
                        alarmName = alarmNameVal
                        nextToken = token
                    },
                )

            response.alarmMuteRuleSummaries?.let { summaries.addAll(it) }
            token = response.nextToken
        } while (token != null)

        summaries.forEach { summary ->
            println("${summary.alarmMuteRuleArn} (${summary.status?.value})")
        }
    }
    return summaries
}
```

- For API details, see ListAlarmMuteRules in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def list_alarm_mute_rules(self, alarm_name=None, statuses=None):
        """
        Lists alarm mute rules in the account.

        :param alarm_name: When specified, only rules that target this alarm are
                           returned.
```

```
        :param statuses: When specified, only rules in these statuses are
returned.
                         Valid values are 'SCHEDULED', 'ACTIVE', and 'EXPIRED'.
        :return: The list of mute rule summaries.
        """
        summaries = []
        try:
            next_token = None
            while True:
                kwargs = {}
                if alarm_name is not None:
                    kwargs["AlarmName"] = alarm_name
                if statuses is not None:
                    kwargs["Statuses"] = statuses
                if next_token is not None:
                    kwargs["NextToken"] = next_token
                response = self.cloudwatch_client.list_alarm_mute_rules(**kwargs)
                summaries.extend(response.get("AlarmMuteRuleSummaries", []))
                next_token = response.get("NextToken")
                if not next_token:
                    break
        except ClientError:
            logger.exception("Couldn't list alarm mute rules.")
            raise
        else:
            logger.info("Got %s alarm mute rules.", len(summaries))
            return summaries
```

- For API details, see ListAlarmMuteRules in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Lists the alarm mute rules in the account, optionally filtered to the rules
 that
# target one alarm.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String, nil] When given, only rules that target this alarm
 are
#   returned.
# @return [Array] The mute rule summaries, as
#   Aws::CloudWatch::Types::AlarmMuteRuleSummary.
def alarm_mute_rules(cloudwatch_client, alarm_name = nil)
  summaries = []
  next_token = nil

  loop do
    response = cloudwatch_client.list_alarm_mute_rules(
      alarm_name: alarm_name,
      next_token: next_token
    )
    summaries.concat(response.alarm_mute_rule_summaries)
    next_token = response.next_token
    break if next_token.nil? || next_token.empty?
  end

  summaries
rescue StandardError => e
  puts "Error listing alarm mute rules: #{e.message}"
  []
end
```

- For API details, see ListAlarmMuteRules in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListDashboards` with an AWS SDK or CLI

The following code examples show how to use `ListDashboards`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Creating a monitoring dashboard with function name as a variable

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Get a list of dashboards.
/// </summary>
/// <returns>A list of DashboardEntry objects.</returns>
public async Task<List<DashboardEntry>> ListDashboards()
{
    var results = new List<DashboardEntry>();
    var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
        new ListDashboardsRequest());
    // Get the entire list using the paginator.
    await foreach (var data in paginateDashboards.DashboardEntries)
    {
        results.Add(data);
    }

    return results;
}
```

- For API details, see ListDashboards in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve a list of Dashboards

The following `list-dashboards` example lists all the Dashboards in the specified account.

```
aws cloudwatch list-dashboards
```

Output:

```
{
    "DashboardEntries": [
        {
            "DashboardName": "Dashboard-A",
            "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/
Dashboard-A",
            "LastModified": "2024-10-11T18:40:11+00:00",
            "Size": 271
        },
        {
            "DashboardName": "Dashboard-B",
            "DashboardArn": "arn:aws:cloudwatch::123456789012:dashboard/
Dashboard-B",
            "LastModified": "2024-10-11T18:44:41+00:00",
            "Size": 522
        }
    ]
}
```

For more information, see Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.

- For API details, see ListDashboards in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Lists the available dashboards.
    *
    * @return a {@link CompletableFuture} that completes when the operation is
finished.
    * The future will complete exceptionally if an error occurs while listing
the dashboards.
    */
   public CompletableFuture<Void> listDashboardsAsync() {
       ListDashboardsRequest listDashboardsRequest =
ListDashboardsRequest.builder().build();
       ListDashboardsPublisher paginator =
getAsyncClient().listDashboardsPaginator(listDashboardsRequest);
       return paginator.subscribe(response -> {
           response.dashboardEntries().forEach(entry -> {
               logger.info("Dashboard name is: {} ", entry.dashboardName());
               logger.info("Dashboard ARN is: {} ", entry.dashboardArn());
           });
       }).exceptionally(ex -> {
           logger.info("Failed to list dashboards: {} ", ex.getMessage());
           throw new RuntimeException("Error occurred while listing dashboards",
ex);
       });
   }
```

- For API details, see ListDashboards in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listDashboards() {
    CloudWatchClient { region = "us-east-1" }.use { cwClient ->
        cwClient
            .listDashboardsPaginated({})
            .transform { it.dashboardEntries?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name is ${obj.dashboardName}")
                println("Dashboard ARN is ${obj.dashboardArn}")
            }
    }
}
```

- For API details, see ListDashboards in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns the collection of dashboards for your account.

```
Get-CWDashboardList
```

Output:

```
DashboardArn DashboardName LastModified        Size
------------ ------------- ------------        ----
arn:...      Dashboard1    7/6/2017 8:14:15 PM 252
```

Example 2: Returns the collection of dashboards for your account whose names start with the prefix 'dev'.

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- For API details, see ListDashboards in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns the collection of dashboards for your account.

```
Get-CWDashboardList
```

Output:

```
DashboardArn DashboardName LastModified        Size
------------ ------------- ------------        ----
arn:...      Dashboard1    7/6/2017 8:14:15 PM 252
```

Example 2: Returns the collection of dashboards for your account whose names start with the prefix 'dev'.

```
Get-CWDashboardList -DashboardNamePrefix dev
```

- For API details, see ListDashboards in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListMetrics` with an AWS SDK or CLI

The following code examples show how to use `ListMetrics`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics

- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// List metrics available, optionally within a namespace.
   /// </summary>
   /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
   /// <param name="filter">Optional dimension filter.</param>
   /// <param name="metricName">Optional metric name filter.</param>
   /// <returns>The list of metrics.</returns>
   public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
   {
       var results = new List<Metric>();
       var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
           new ListMetricsRequest
           {
               Namespace = metricNamespace,
               Dimensions = filter != null ? new List<DimensionFilter>
{ filter } : null,
               MetricName = metricName
           });
       // Get the entire list using the paginator.
       await foreach (var metric in paginateMetrics.Metrics)
       {
           results.Add(metric);
       }

       return results;
   }
```

- For API details, see ListMetrics in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/ListMetricsRequest.h>
#include <aws/monitoring/model/ListMetricsResult.h>
#include <iomanip>
#include <iostream>
```

List the metrics.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::ListMetricsRequest request;

        if (argc > 1)
        {
            request.SetMetricName(argv[1]);
        }

        if (argc > 2)
        {
            request.SetNamespace(argv[2]);
        }

        bool done = false;
        bool header = false;
        while (!done)
        {
```

```
            auto outcome = cw.ListMetrics(request);
            if (!outcome.IsSuccess())
            {
                std::cout << "Failed to list CloudWatch metrics:" <<
                    outcome.GetError().GetMessage() << std::endl;
                break;
            }

            if (!header)
            {
                std::cout << std::left << std::setw(48) << "MetricName" <<
                    std::setw(32) << "Namespace" << "DimensionNameValuePairs" <<
                    std::endl;
                header = true;
            }

            const auto &metrics = outcome.GetResult().GetMetrics();
            for (const auto &metric : metrics)
            {
                std::cout << std::left << std::setw(48) <<
                    metric.GetMetricName() << std::setw(32) <<
                    metric.GetNamespace();
                const auto &dimensions = metric.GetDimensions();
                for (auto iter = dimensions.cbegin();
                    iter != dimensions.cend(); ++iter)
                {
                    const auto &dimkv = *iter;
                    std::cout << dimkv.GetName() << " = " << dimkv.GetValue();
                    if (iter + 1 != dimensions.cend())
                    {
                        std::cout << ", ";
                    }
                }
                std::cout << std::endl;
            }

            const auto &next_token = outcome.GetResult().GetNextToken();
            request.SetNextToken(next_token);
            done = next_token.empty();
        }
```

- For API details, see ListMetrics in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list the metrics for Amazon SNS

The following `list-metrics` example displays the metrics for Amazon SNS.

```
aws cloudwatch list-metrics \
    --namespace "AWS/SNS"
```

Output:

```
{
    "Metrics": [
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "PublishSize"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
```

```
            "MetricName": "NumberOfNotificationsFailed"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "NotifyMe"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfMessagesPublished"
        },
        {
            "Namespace": "AWS/SNS",
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsDelivered"
        },
        {
            "Namespace": "AWS/SNS",
```

```
            "Dimensions": [
                {
                    "Name": "TopicName",
                    "Value": "CFO"
                }
            ],
            "MetricName": "NumberOfNotificationsFailed"
        }
    ]
}
```

For more information, see View available metrics in the *Amazon CloudWatch User Guide*.

- For API details, see ListMetrics in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Retrieves a list of metric names for the specified namespace.
    *
    * @param namespace the namespace for which to retrieve the metric names
    * @return a {@link CompletableFuture} that, when completed, contains an
{@link ArrayList} of
    * the metric names in the specified namespace
    * @throws RuntimeException if an error occurs while listing the metrics
    */
   public CompletableFuture<ArrayList<String>> listMetsAsync(String namespace) {
       ListMetricsRequest request = ListMetricsRequest.builder()
           .namespace(namespace)
           .build();

       ListMetricsPublisher metricsPaginator =
getAsyncClient().listMetricsPaginator(request);
       Set<String> metSet = new HashSet<>();
```

```
        CompletableFuture<Void> future = metricsPaginator.subscribe(response -> {
            response.metrics().forEach(metric -> {
                String metricName = metric.metricName();
                metSet.add(metricName);
            });
        });

        return future
            .thenApply(ignored -> new ArrayList<>(metSet))
            .exceptionally(exception -> {
                throw new RuntimeException("Failed to list metrics: " +
 exception.getMessage(), exception);
            });
    }
```

- For API details, see ListMetrics in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = async () => {
  // Use the AWS console to see available namespaces and metric names. Custom
 metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
```

```
    Dimensions: [
      {
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListMetrics in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ListMetrics in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listMets(namespaceVal: String?): ArrayList<String>? {
    val metList = ArrayList<String>()
```

```
    val request =
        ListMetricsRequest {
            namespace = namespaceVal
        }
    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val reponse = cwClient.listMetrics(request)
        reponse.metrics?.forEach { metrics ->
            val data = metrics.metricName
            if (!metList.contains(data)) {
                metList.add(data!!)
            }
        }
    }
    return metList
}
```

- For API details, see ListMetrics in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def list_metrics(self, namespace, name, recent=False):
        """
        Gets the metrics within a namespace that have the specified name.
```

```
        If the metric has no dimensions, a single metric is returned.
        Otherwise, metrics for all dimensions are returned.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param recent: When True, only metrics that have been active in the last
                       three hours are returned.
        :return: An iterator that yields the retrieved metrics.
        """
        try:
            kwargs = {"Namespace": namespace, "MetricName": name}
            if recent:
                kwargs["RecentlyActive"] = "PT3H"  # List past 3 hours only
            metric_iter = self.cloudwatch_resource.metrics.filter(**kwargs)
            logger.info("Got metrics for %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't get metrics for %s.%s.", namespace, name)
            raise
        else:
            return metric_iter
```

- For API details, see ListMetrics in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Lists available metrics for a metric namespace in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric.
# @example
#   list_metrics_for_namespace(
```

```
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC'
#   )
def list_metrics_for_namespace(cloudwatch_client, metric_namespace)
  response = cloudwatch_client.list_metrics(namespace: metric_namespace)

  if response.metrics.count.positive?
    response.metrics.each do |metric|
      puts "  Metric name: #{metric.metric_name}"
      if metric.dimensions.count.positive?
        puts '    Dimensions:'
        metric.dimensions.each do |dimension|
          puts "      Name: #{dimension.name}, Value: #{dimension.value}"
        end
      else
        puts 'No dimensions found.'
      end
    end
  else
    puts "No metrics found for namespace '#{metric_namespace}'. " \
      'Note that it could take up to 15 minutes for recently-added metrics ' \
      'to become available.'
  end
end

# Example usage:
def run_me
  metric_namespace = 'SITE/TRAFFIC'
  # Replace us-west-2 with the AWS Region you're using for Amazon CloudWatch.
  region = 'us-east-1'

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)

  # Add three datapoints.
  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'UniqueVisitors',
    'SiteName',
    'example.com',
    5_885.0,
    'Count'
  )
```

```
  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'UniqueVisits',
    'SiteName',
    'example.com',
    8_628.0,
    'Count'
  )

  puts 'Continuing...' unless datapoint_added_to_metric?(
    cloudwatch_client,
    metric_namespace,
    'PageViews',
    'PageURL',
    'example.html',
    18_057.0,
    'Count'
  )

  puts "Metrics for namespace '#{metric_namespace}':"
  list_metrics_for_namespace(cloudwatch_client, metric_namespace)
end

run_me if $PROGRAM_NAME == __FILE__
```

- For API details, see ListMetrics in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   "The following list-metrics example displays the metrics for Amazon
 CloudWatch."
   TRY.
```

```
        oo_result = lo_cwt->listmetrics(            " oo_result is returned for
testing purposes. "
          iv_namespace = iv_namespace ).
        DATA(lt_metrics) = oo_result->get_metrics( ).
        MESSAGE 'Metrics retrieved.' TYPE 'I'.
      CATCH /aws1/cx_cwtinvparamvalueex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListMetrics in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutAlarmMuteRule` with an AWS SDK

The following code examples show how to use `PutAlarmMuteRule`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create or update an alarm mute rule. While a mute rule is active the
targeted
```

```
    /// alarms keep evaluating and keep transitioning between states, but their
 configured
    /// actions do not fire. This is the supported way to suppress notifications
 during a
    /// known maintenance window, instead of disabling alarm actions and relying
 on
    /// someone to turn them back on.
    /// </summary>
    /// <param name="name">The name of the mute rule.</param>
    /// <param name="expression">When the rule activates. For a recurring window,
 use a
    /// five-field cron expression, cron(Minutes Hours Day-of-month Month Day-of-
week),
    /// such as cron(0 2 * * SUN) for every Sunday at 2:00 AM. Note that this is
 five
    /// fields, not the six that Amazon EventBridge uses. For a one-time window,
 use
    /// at(yyyy-MM-ddThh:mm), such as at(2026-09-05T02:00).</param>
    /// <param name="duration">How long the mute window lasts once it activates,
 in
    /// ISO 8601 duration format, from PT1M (one minute) to P15D (15 days). For
 example,
    /// PT2H is two hours and P2DT12H is two days and 12 hours.</param>
    /// <param name="timezone">The time zone the expression is evaluated in, such
 as
    /// America/Los_Angeles.</param>
    /// <param name="alarmNames">The names of up to 100 alarms to mute. If null
 or empty,
    /// the rule applies to all alarms in the account.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAlarmMuteRule(string name, string expression,
 string duration,
        string timezone, List<string>? alarmNames = null)
    {
        var request = new PutAlarmMuteRuleRequest
        {
            Name = name,
            Description = "A mute rule created by the AWS SDK for .NET example.",
            Rule = new Rule
            {
                Schedule = new Schedule
                {
                    Expression = expression,
                    Duration = duration,
```

```
                    Timezone = timezone
                }
            }
        };

        if (alarmNames != null && alarmNames.Any())
        {
            request.MuteTargets = new MuteTargets { AlarmNames = alarmNames };
        }

        var response = await _amazonCloudWatch.PutAlarmMuteRuleAsync(request);

        _logger.LogInformation($"Put alarm mute rule {name}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see PutAlarmMuteRule in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/MuteTargets.h>
#include <aws/monitoring/model/PutAlarmMuteRuleRequest.h>
#include <aws/monitoring/model/Rule.h>
#include <aws/monitoring/model/Schedule.h>
#include <iostream>
```

Create the alarm mute rule.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        // For a recurring window, use a five-field cron expression,
        // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
five
        // fields, not the six that Amazon EventBridge uses. For a one-time
window, use
        // an at expression such as at(2026-09-05T02:00).
        Aws::CloudWatch::Model::Schedule schedule;
        schedule.SetExpression("cron(0 2 * * SUN)");
        // The duration is in ISO 8601 duration format, from PT1M (one minute) to
        // P15D (15 days).
        schedule.SetDuration("PT2H");
        schedule.SetTimezone("America/Los_Angeles");

        Aws::CloudWatch::Model::Rule rule;
        rule.SetSchedule(schedule);

        // Target up to 100 alarms. If MuteTargets is not set, the rule applies
to every
        // alarm in the account.
        Aws::CloudWatch::Model::MuteTargets muteTargets;
        muteTargets.AddAlarmNames(alarm_name);

        Aws::CloudWatch::Model::PutAlarmMuteRuleRequest request;
        request.SetName(mute_rule_name);
        request.SetDescription("A mute rule created by the AWS SDK for C++.");
        request.SetRule(rule);
        request.SetMuteTargets(muteTargets);

        auto outcome = cw.PutAlarmMuteRule(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to put alarm mute rule: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            std::cout << "Successfully put alarm mute rule " << mute_rule_name
                      << std::endl;
        }
```

- For API details, see PutAlarmMuteRule in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Creates or updates an alarm mute rule. While a mute rule is active the
 targeted
     * alarms keep evaluating and keep transitioning between states, but their
 configured
     * actions do not fire. This is the supported way to suppress notifications
 during a
     * known maintenance window instead of disabling alarm actions and hoping
 someone
     * remembers to turn them back on.
     *
     * @param cw         the CloudWatch client
     * @param name       the name of the mute rule
     * @param expression when the rule activates. For a recurring window, use a
     *                   five-field cron expression,
     *                   {@code cron(Minutes Hours Day-of-month Month Day-of-
week)},
     *                   such as {@code cron(0 2 * * SUN)} for every Sunday at
 2:00 AM.
     *                   Note that this is five fields, not the six that Amazon
     *                   EventBridge uses. For a one-time window, use
     *                   {@code at(yyyy-MM-ddThh:mm)}, such as
     *                   {@code at(2026-09-05T02:00)}.
     * @param duration   how long the mute window lasts once it activates, in ISO
 8601
     *                   duration format, from {@code PT1M} (one minute) to
     *                   {@code P15D} (15 days). For example, {@code PT2H} is two
 hours
     *                   and {@code P2DT12H} is two days and 12 hours.
```

```
    * @param timezone   the time zone the expression is evaluated in, such as
    *                   {@code America/Los_Angeles}
    * @param alarmNames the names of up to 100 alarms to mute. If empty, the
rule
    *                   applies to all alarms in the account.
    */
   public static void putAlarmMuteRule(CloudWatchClient cw, String name, String
expression, String duration,
           String timezone, List<String> alarmNames) {
       try {
           Schedule schedule = Schedule.builder()
                   .expression(expression)
                   .duration(duration)
                   .timezone(timezone)
                   .build();

           PutAlarmMuteRuleRequest.Builder request =
PutAlarmMuteRuleRequest.builder()
                   .name(name)
                   .description("Mute rule created by the AWS SDK for Java 2.x
example.")
                   .rule(Rule.builder().schedule(schedule).build());

           if (alarmNames != null && !alarmNames.isEmpty()) {

request.muteTargets(MuteTargets.builder().alarmNames(alarmNames).build());
           }

           cw.putAlarmMuteRule(request.build());
           System.out.printf("Put alarm mute rule %s.%n", name);

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see PutAlarmMuteRule in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { PutAlarmMuteRuleCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Create or update an alarm mute rule. While a mute rule is active the targeted
 alarms
// keep evaluating and keep transitioning between states, but their configured
 actions
// do not fire. This is the supported way to suppress notifications during a
 known
// maintenance window, instead of disabling alarm actions and hoping someone
 remembers
// to turn them back on.
const run = async () => {
  const command = new PutAlarmMuteRuleCommand({
    Name: process.env.CLOUDWATCH_MUTE_RULE_NAME, // Set CLOUDWATCH_MUTE_RULE_NAME
 to the name of the mute rule.
    Description: "Suppress checkout CPU pages during Sunday patching.",
    Rule: {
      Schedule: {
        // For a recurring window, use a five-field cron expression,
        // cron(Minutes Hours Day-of-month Month Day-of-week). Note that this is
 five
        // fields, not the six that Amazon EventBridge uses. For a one-time
 window, use
        // an at expression such as "at(2026-09-05T02:00)".
        Expression: "cron(0 2 * * SUN)",
        // How long the mute window lasts once it activates, in ISO 8601 duration
        // format, from PT1M (one minute) to P15D (15 days).
        Duration: "PT2H",
        Timezone: "America/Los_Angeles",
      },
    },
```

```
    // Target up to 100 alarms by name. Omit MuteTargets to mute every alarm in
 the
    // account.
    MuteTargets: {
      AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    },
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see PutAlarmMuteRule in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun putAlarmMuteRule(
    muteRuleName: String,
    expressionVal: String,
    durationVal: String,
    alarmNamesVal: List<String>,
    timezoneVal: String = "America/Los_Angeles",
) {
    // For a recurring window, use a five-field cron expression,
    // cron(Minutes Hours Day-of-month Month Day-of-week), such as cron(0 2 * *
 SUN).
```

```
    // Note that this is five fields, not the six that Amazon EventBridge uses.
 For a
    // one-time window, use at(yyyy-MM-ddThh:mm), such as at(2026-09-05T02:00).
 The
    // duration is in ISO 8601 duration format, from PT1M (one minute) to P15D
 (15 days).
    val scheduleOb =
        Schedule {
            expression = expressionVal
            duration = durationVal
            timezone = timezoneVal
        }

    val request =
        PutAlarmMuteRuleRequest {
            name = muteRuleName
            description = "A mute rule created by the Kotlin SDK"
            rule =
                Rule {
                    schedule = scheduleOb
                }
            // Target up to 100 alarms. If muteTargets is omitted, the rule
 applies to
            // every alarm in the account.
            if (alarmNamesVal.isNotEmpty()) {
                muteTargets =
                    MuteTargets {
                        alarmNames = alarmNamesVal
                    }
            }
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putAlarmMuteRule(request)
        println("Successfully put alarm mute rule $muteRuleName")
    }
}
```

- For API details, see PutAlarmMuteRule in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def put_alarm_mute_rule(
        self,
        name,
        expression,
        duration,
        alarm_names=None,
        timezone=None,
        description=None,
    ):
        """
```

```
        Creates or updates an alarm mute rule. While a mute rule is active the
 targeted
        alarms keep evaluating and keep transitioning between states, but their
        configured actions do not fire. This is the supported way to suppress
        notifications during a known maintenance window instead of disabling
 alarm
        actions and hoping someone remembers to turn them back on.

        :param name: The name of the mute rule.
        :param expression: When the rule activates. For a recurring window, use a
                           five-field cron expression,
                           'cron(Minutes Hours Day-of-month Month Day-of-week)',
 such as
                           'cron(0 2 * * SUN)' for every Sunday at 2:00 AM. Note
 that
                           this is five fields, not the six that Amazon
 EventBridge
                           uses. For a one-time window, use 'at(yyyy-MM-
ddThh:mm)',
                           such as 'at(2026-09-05T02:00)'.
        :param duration: How long the mute window lasts once it activates, in ISO
 8601
                         duration format, from 'PT1M' (one minute) to 'P15D' (15
 days).
                         For example, 'PT2H' is two hours and 'P2DT12H' is two
 days and
                         12 hours.
        :param alarm_names: The names of up to 100 alarms to mute. If omitted,
 the rule
                            applies to all alarms in the account.
        :param timezone: The time zone the expression is evaluated in, such as
                         'America/Los_Angeles'.
        :param description: The description of the mute rule.
        """
        schedule = {"Expression": expression, "Duration": duration}
        if timezone is not None:
            schedule["Timezone"] = timezone

        kwargs = {"Name": name, "Rule": {"Schedule": schedule}}
        if alarm_names is not None:
            kwargs["MuteTargets"] = {"AlarmNames": alarm_names}
        if description is not None:
            kwargs["Description"] = description
```

```
        try:
            self.cloudwatch_client.put_alarm_mute_rule(**kwargs)
            logger.info("Put alarm mute rule %s.", name)
        except ClientError:
            logger.exception("Couldn't put alarm mute rule %s.", name)
            raise
```

- For API details, see PutAlarmMuteRule in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Creates or updates an alarm mute rule. While a mute rule is active the targeted
 alarms
# keep evaluating and keep transitioning between states, but their configured
 actions do
# not fire. This is the supported way to suppress notifications during a known
# maintenance window, instead of disabling alarm actions and relying on someone
 to turn
# them back on.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param name [String] The name of the mute rule.
# @param schedule [Hash] The mute window, mirroring the +rule.schedule+ shape:
#   * +:expression+ [String] When the rule activates. For a recurring window, use
 a
#     five-field cron expression,
#     'cron(Minutes Hours Day-of-month Month Day-of-week)', such as
#     'cron(0 2 * * SUN)' for every Sunday at 2:00 AM. Note that this is five
 fields,
#     not the six that Amazon EventBridge uses. For a one-time window, use
#     'at(yyyy-MM-ddThh:mm)', such as 'at(2026-09-05T02:00)'.
```

```
#   * +:duration+ [String] How long the mute window lasts once it activates, in
 ISO 8601
#     duration format, from 'PT1M' (one minute) to 'P15D' (15 days). For example,
#     'PT2H' is two hours and 'P2DT12H' is two days and 12 hours.
#   * +:timezone+ [String] The time zone the expression is evaluated in, such as
#     'America/Los_Angeles'.
# @param alarm_names [Array] The names of up to 100 alarms to mute. If empty, the
 rule
#   applies to all alarms in the account.
# @param description [String] A description of the mute rule.
# @return [Boolean] true if the mute rule was created or updated; otherwise,
 false.
def alarm_mute_rule_created_or_updated?(
  cloudwatch_client,
  name,
  schedule,
  alarm_names,
  description
)
  params = {
    name: name,
    description: description,
    rule: {
      schedule: {
        expression: schedule[:expression],
        duration: schedule[:duration],
        timezone: schedule[:timezone]
      }
    }
  }
  params[:mute_targets] = { alarm_names: alarm_names } unless alarm_names.empty?

  cloudwatch_client.put_alarm_mute_rule(params)
  true
rescue StandardError => e
  puts "Error putting alarm mute rule: #{e.message}"
  false
end
```

- For API details, see PutAlarmMuteRule in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutAnomalyDetector` with an AWS SDK or CLI

The following code examples show how to use `PutAnomalyDetector`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutAnomalyDetector in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To create an anomaly detection model

The following `put-anomaly-detector` example creates an anomaly detection model for a CloudWatch metric.

```
aws cloudwatch put-anomaly-detector \
    --namespace AWS/Logs \
    --metric-name IncomingBytes \
    --stat SampleCount
```

This command produces no output.

For more information, see Using CloudWatch anomaly detection in the *Amazon CloudWatch User Guide*.

- For API details, see PutAnomalyDetector in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Adds an anomaly detector for the given file.
    *
    * @param fileName the name of the file containing the anomaly detector
configuration
    * @return a {@link CompletableFuture} that completes when the anomaly
detector has been added
    */
   public CompletableFuture<Void> addAnomalyDetectorAsync(String fileName) {
```

```
        CompletableFuture<JsonNode> readFileFuture =
CompletableFuture.supplyAsync(() -> {
            try {
                JsonParser parser = new JsonFactory().createParser(new
File(fileName));
                return new ObjectMapper().readTree(parser); // Return the root
node
            } catch (IOException e) {
                throw new RuntimeException("Failed to read or parse the file",
e);
            }
        });

        return readFileFuture.thenCompose(rootNode -> {
            try {
                String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
                String customMetricName =
rootNode.findValue("customMetricName").asText();

                SingleMetricAnomalyDetector singleMetricAnomalyDetector =
SingleMetricAnomalyDetector.builder()
                    .metricName(customMetricName)
                    .namespace(customMetricNamespace)
                    .stat("Maximum")
                    .build();

                PutAnomalyDetectorRequest anomalyDetectorRequest =
PutAnomalyDetectorRequest.builder()
                    .singleMetricAnomalyDetector(singleMetricAnomalyDetector)
                    .build();

                return
getAsyncClient().putAnomalyDetector(anomalyDetectorRequest).thenAccept(response
-> {
                    logger.info("Added anomaly detector for metric {}",
customMetricName);
                });
            } catch (Exception e) {
                throw new RuntimeException("Failed to create anomaly detector",
e);
            }
        }).whenComplete((result, exception) -> {
            if (exception != null) {
```

```
                throw new RuntimeException("Error adding anomaly detector",
 exception);
            }
        });
    }
```

- For API details, see PutAnomalyDetector in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun addAnomalyDetector(
    namespaceVal: String,
    metricNameVal: String,
) {
    val singleMetricAnomalyDetectorVal =
        SingleMetricAnomalyDetector {
            namespace = namespaceVal
            metricName = metricNameVal
            stat = "Maximum"
        }

    val request =
        PutAnomalyDetectorRequest {
            singleMetricAnomalyDetector = singleMetricAnomalyDetectorVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putAnomalyDetector(request)
        println("Added an anomaly detector for metric $metricNameVal.")
    }
}
```

- For API details, see PutAnomalyDetector in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutDashboard` with an AWS SDK or CLI

The following code examples show how to use `PutDashboard`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Creating a monitoring dashboard with function name as a variable
- Using property variables in monitoring dashboards to monitor multiple serverless functions

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
```

```
        // Best practice is to include a text widget indicating this dashboard
 was created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }
```

- For API details, see PutDashboard in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To create a dashboard

The following `put-dashboard` example creates a dashboard named `Dashboard-A` in the specified account.

```
aws cloudwatch put-dashboard \
    --dashboard-name Dashboard-A \
    --dashboard-body '{"widgets":
[{"height":6,"width":6,"y":0,"x":0,"type":"metric","properties":
{"view":"timeSeries","stacked":false,"metrics":
[["Namespace","CPUUtilization","Environment","Prod","Type","App"]],"region":"us-
west-2"}}]}'
```

Output:

```
{
    "DashboardValidationMessages": []
}
```

For more information, see Creating a CloudWatch dashboard in the *Amazon CloudWatch User Guide*.

- For API details, see PutDashboard in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Creates a new dashboard with the specified name and body.
    *
    * @param dashboardName the name of the dashboard to be created
    * @param dashboardBody the dashboard body, as JSON
    * @return a {@link CompletableFuture} representing the asynchronous
operation of creating the dashboard
    */
   public CompletableFuture<PutDashboardResponse> createDashboardAsync(String
dashboardName, String dashboardBody) {
       PutDashboardRequest dashboardRequest = PutDashboardRequest.builder()
           .dashboardName(dashboardName)
           .dashboardBody(dashboardBody)
           .build();

       return getAsyncClient().putDashboard(dashboardRequest)
           .handle((response, ex) -> {
               if (ex != null) {
                   logger.info("Failed to create dashboard: {}",
ex.getMessage());
                   throw new RuntimeException("Dashboard creation failed", ex);
               } else {
                   // Handle the normal response case
                   logger.info("{} was successfully created.", dashboardName);
                   List<DashboardValidationMessage> messages =
response.dashboardValidationMessages();
                   if (messages.isEmpty()) {
                       logger.info("There are no messages in the new
Dashboard.");
```

```
                    } else {
                        for (DashboardValidationMessage message : messages) {
                            logger.info("Message: {}", message.message());
                        }
                    }
                    return response; // Return the response for further use
                }
            });
    }
```

- For API details, see PutDashboard in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createDashboard(
    dashboardNameVal: String,
    dashboardBodyVal: String,
) {
    val dashboardRequest =
        PutDashboardRequest {
            dashboardName = dashboardNameVal
            dashboardBody = dashboardBodyVal
        }

    CloudWatchClient.fromEnvironment { region = REGION }.use { cwClient ->
        val response = cwClient.putDashboard(dashboardRequest)
        println("$dashboardNameVal was successfully created.")
        val messages = response.dashboardValidationMessages
        if (messages != null) {
            if (messages.isEmpty()) {
                println("There are no messages in the new Dashboard")
            } else {
                for (message in messages) {
```

```
                println("Message is: ${message.message}")
            }
        }
    }
}
```

- For API details, see PutDashboard in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: Creates or updates the dashboard named 'Dashboard1' to include two metric widgets side by side.

```
$dashBody = @"
{
    "widgets":[
        {
             "type":"metric",
             "x":0,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/EC2",
                      "CPUUtilization",
                      "InstanceId",
                      "i-012345"
                   ]
                ],
                "period":300,
                "stat":"Average",
                "region":"us-east-1",
                "title":"EC2 Instance CPU"
             }
        },
        {
             "type":"metric",
```

```
            "x":12,
            "y":0,
            "width":12,
            "height":6,
            "properties":{
               "metrics":[
                  [
                     "AWS/S3",
                     "BucketSizeBytes",
                     "BucketName",
                     "amzn-s3-demo-bucket"
                  ]
               ],
               "period":86400,
               "stat":"Maximum",
               "region":"us-east-1",
               "title":"amzn-s3-demo-bucket bytes"
            }
         }
    ]
}
"@

Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

Example 2: Creates or updates the dashboard, piping the content describing the dashboard into the cmdlet.

```
$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- For API details, see PutDashboard in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Creates or updates the dashboard named 'Dashboard1' to include two metric widgets side by side.

```
$dashBody = @"
{
    "widgets":[
        {
             "type":"metric",
             "x":0,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/EC2",
                      "CPUUtilization",
                      "InstanceId",
                      "i-012345"
                   ]
                ],
                "period":300,
                "stat":"Average",
                "region":"us-east-1",
                "title":"EC2 Instance CPU"
             }
        },
        {
             "type":"metric",
             "x":12,
             "y":0,
             "width":12,
             "height":6,
             "properties":{
                "metrics":[
                   [
                      "AWS/S3",
                      "BucketSizeBytes",
                      "BucketName",
                      "amzn-s3-demo-bucket"
                   ]
                ],
                "period":86400,
                "stat":"Maximum",
                "region":"us-east-1",
                "title":"amzn-s3-demo-bucket bytes"
```

```
            }
        }
    ]
}
"@

Write-CWDashboard -DashboardName Dashboard1 -DashboardBody $dashBody
```

Example 2: Creates or updates the dashboard, piping the content describing the dashboard into the cmdlet.

```
$dashBody = @"
{
...
}
"@

$dashBody | Write-CWDashboard -DashboardName Dashboard1
```

- For API details, see PutDashboard in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutMetricAlarm` with an AWS SDK or CLI

The following code examples show how to use `PutMetricAlarm`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Get started with alarms
- Manage custom metrics and alarms
- Run CPU stress tests on virtual machine instances using fault injection
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
    /// <summary>
    /// Create an alarm that evaluates a PromQL query.
    ///
    /// A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
    /// match many series at once, and each matching series is tracked separately
as a
    /// contributor. Instead of counting breaching periods, you specify
durations: a
    /// contributor moves to ALARM after it breaches continuously for the pending
period,
    /// and back to OK after it stops breaching for the recovery period. A PromQL
alarm
    /// starts in the OK state rather than INSUFFICIENT_DATA.
    ///
    /// EvaluationCriteria is a union and is mutually exclusive with the classic
    /// MetricName and Metrics properties. When you use it you must also set
    /// EvaluationInterval, and you must not set Period, Statistic, Threshold,
    /// ComparisonOperator, EvaluationPeriods, DatapointsToAlarm, or
TreatMissingData.
    /// </summary>
    /// <param name="alarmName">The name of the alarm, unique within the
Region.</param>
    /// <param name="query">The PromQL query to evaluate, such as
    /// avg(cpu_utilization_percent) &gt; 80. The comparison belongs in the query
itself;
    /// there is no separate threshold property.</param>
    /// <param name="evaluationInterval">How often, in seconds, to run the query.
Valid
    /// values are 10, 20, 30, and any multiple of 60, up to 3600.</param>
```

```
   /// <param name="pendingPeriod">How long, in seconds, a contributor must
breach
   /// continuously before it moves to ALARM.</param>
   /// <param name="recoveryPeriod">How long, in seconds, a contributor must
stop
   /// breaching before it moves back to OK.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutPromQLMetricAlarm(string alarmName, string query,
       int evaluationInterval = 60, int pendingPeriod = 300, int recoveryPeriod
= 120)
   {
       var response = await _amazonCloudWatch.PutMetricAlarmAsync(
           new PutMetricAlarmRequest
           {
               AlarmName = alarmName,
               AlarmDescription = "A PromQL alarm created by the AWS SDK
for .NET example.",
               EvaluationCriteria = new EvaluationCriteria
               {
                   PromQLCriteria = new AlarmPromQLCriteria
                   {
                       Query = query,
                       PendingPeriod = pendingPeriod,
                       RecoveryPeriod = recoveryPeriod
                   }
               },
               EvaluationInterval = evaluationInterval
           });

       _logger.LogInformation($"Created PromQL alarm {alarmName} for query
{query}.");
       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }
```

Create an alarm that evaluates a single CloudWatch metric.

```
   /// <summary>
   /// Add a metric alarm to send an email when the metric passes a threshold.
   /// </summary>
   /// <param name="alarmDescription">A description of the alarm.</param>
   /// <param name="alarmName">The name for the alarm.</param>
   /// <param name="comparison">The type of comparison to use.</param>
```

```
   /// <param name="metricName">The name of the metric for the alarm.</param>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="threshold">The threshold value for the alarm.</param>
   /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
       string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
   {
       try
       {
           var putEmailAlarmResponse = await
_amazonCloudWatch.PutMetricAlarmAsync(
               new PutMetricAlarmRequest()
               {
                   AlarmActions = alarmActions,
                   AlarmDescription = alarmDescription,
                   AlarmName = alarmName,
                   ComparisonOperator = comparison,
                   Threshold = threshold,
                   Namespace = metricNamespace,
                   MetricName = metricName,
                   EvaluationPeriods = 1,
                   Period = 10,
                   Statistic = new Statistic("Maximum"),
                   DatapointsToAlarm = 1,
                   TreatMissingData = "ignore"
               });
           return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (LimitExceededException lex)
       {
           _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota
has already been reached.");
       }

       return false;
   }

   /// <summary>
   /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
```

```
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
 topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to
 append to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:
{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- For API details, see PutMetricAlarm in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files for a PromQL alarm.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/AlarmPromQLCriteria.h>
#include <aws/monitoring/model/EvaluationCriteria.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::AlarmPromQLCriteria promQLCriteria;
        promQLCriteria.SetQuery(query);
        // A contributor moves to ALARM after breaching continuously for 300
 seconds,
        // and back to OK after 120 seconds without breaching.
        promQLCriteria.SetPendingPeriod(300);
        promQLCriteria.SetRecoveryPeriod(120);

        Aws::CloudWatch::Model::EvaluationCriteria evaluationCriteria;
        evaluationCriteria.SetPromQLCriteria(promQLCriteria);

        Aws::CloudWatch::Model::PutMetricAlarmRequest request;
        request.SetAlarmName(alarm_name);
        request.SetAlarmDescription("A PromQL alarm created by the AWS SDK for C+
+.");
        request.SetEvaluationCriteria(evaluationCriteria);
        // Valid values are 10, 20, 30, and any multiple of 60, up to 3600.
        request.SetEvaluationInterval(30);

        auto outcome = cw.PutMetricAlarm(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to create PromQL alarm: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            std::cout << "Successfully created PromQL alarm " << alarm_name
                      << " for query " << query << std::endl;
        }
```

Include the required files for a metric alarm.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricAlarmRequest.h>
#include <iostream>
```

Create the alarm to watch the metric.

```
        Aws::CloudWatch::CloudWatchClient cw;
        Aws::CloudWatch::Model::PutMetricAlarmRequest request;
        request.SetAlarmName(alarm_name);
        request.SetComparisonOperator(
            Aws::CloudWatch::Model::ComparisonOperator::GreaterThanThreshold);
        request.SetEvaluationPeriods(1);
        request.SetMetricName("CPUUtilization");
        request.SetNamespace("AWS/EC2");
        request.SetPeriod(60);
        request.SetStatistic(Aws::CloudWatch::Model::Statistic::Average);
        request.SetThreshold(70.0);
        request.SetActionsEnabled(false);
        request.SetAlarmDescription("Alarm when server CPU exceeds 70%");
        request.SetUnit(Aws::CloudWatch::Model::StandardUnit::Seconds);

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("InstanceId");
        dimension.SetValue(instanceId);

        request.AddDimensions(dimension);

        auto outcome = cw.PutMetricAlarm(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to create CloudWatch alarm:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully created CloudWatch alarm " << alarm_name
                << std::endl;
        }
```

- For API details, see PutMetricAlarm in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To send an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent

The following `put-metric-alarm` example sends an Amazon Simple Notification Service email message when CPU utilization exceeds 70 percent.

```
aws cloudwatch put-metric-alarm \
    --alarm-name cpu-mon \
    --alarm-description "Alarm when CPU exceeds 70 percent" \
    --metric-name CPUUtilization \
    --namespace AWS/EC2 \
    --statistic Average \
    --period 300 \
    --threshold 70 \
    --comparison-operator GreaterThanThreshold \
    --dimensions "Name=InstanceId,Value=i-12345678" \
    --evaluation-periods 2 \
    --alarm-actions arn:aws:sns:us-west-2:123456789012:MyTopic \
    --unit Percent
```

This command produces no output.

If an alarm with the same name already exists, it will be overwritten by the new alarm.

For more information, see Create a CPU usage alarm in the *Amazon CloudWatch User Guide*.

Example 2: To specify multiple dimensions

The following `put-metric-alarm` example illustrates how to specify multiple dimensions. Each dimension is specified as a Name/Value pair, with a comma between the name and the value. Multiple dimensions are separated by a space.

```
aws cloudwatch put-metric-alarm \
    --alarm-name "Default_Test_Alarm3" \
    --alarm-description "The default example alarm" \
    --namespace "CW EXAMPLE METRICS" \
    --metric-name Default_Test \
    --statistic Average \
    --period 60 \
```

```
    --evaluation-periods 3 \
    --threshold 50 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --dimensions Name=key1,Value=value1 Name=key2,Value=value2
```

This command produces no output.

For more information, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

- For API details, see PutMetricAlarm in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
   /**
    * Creates an alarm that evaluates a PromQL query.
    *
    * <p>A PromQL alarm differs from a classic metric alarm in a few ways. The
query
    * can match many series at once, and each matching series is tracked
separately as
    * a contributor. Instead of counting breaching periods, you specify
durations: a
    * contributor moves to ALARM after it breaches continuously for the pending
period,
    * and back to OK after it stops breaching for the recovery period. A PromQL
alarm
    * starts in the OK state rather than INSUFFICIENT_DATA.
    *
    * <p>{@link EvaluationCriteria} is a union and is mutually exclusive with
the
```

```
    * classic {@code metricName} and {@code metrics} parameters. When you use it
you
    * must also set {@code evaluationInterval}, and you must not set {@code
period},
    * {@code statistic}, {@code threshold}, {@code comparisonOperator},
    * {@code evaluationPeriods}, {@code datapointsToAlarm}, or
    * {@code treatMissingData}.
    *
    * @param cw                 the CloudWatch client
    * @param alarmName          the name of the alarm, unique within the Region
    * @param query              the PromQL query to evaluate, such as
    *                           {@code avg(cpu_utilization_percent) > 80}. The
    *                           comparison belongs in the query itself; there is
no
    *                           separate threshold parameter.
    * @param evaluationInterval how often, in seconds, to run the query. Valid
values
    *                           are 10, 20, 30, and any multiple of 60, up to
3600.
    * @param pendingPeriod      how long, in seconds, a contributor must breach
    *                           continuously before it moves to ALARM
    * @param recoveryPeriod     how long, in seconds, a contributor must stop
breaching
    *                           before it moves back to OK
    */
   public static void putPromQLMetricAlarm(CloudWatchClient cw, String
alarmName, String query,
           int evaluationInterval, int pendingPeriod, int recoveryPeriod) {
       try {
           AlarmPromQLCriteria promQLCriteria = AlarmPromQLCriteria.builder()
                   .query(query)
                   .pendingPeriod(pendingPeriod)
                   .recoveryPeriod(recoveryPeriod)
                   .build();

           PutMetricAlarmRequest request = PutMetricAlarmRequest.builder()
                   .alarmName(alarmName)
                   .alarmDescription("PromQL alarm created by the AWS SDK for
Java 2.x example.")
                   .evaluationCriteria(EvaluationCriteria.builder()
                           .promQLCriteria(promQLCriteria)
                           .build())
                   .evaluationInterval(evaluationInterval)
                   .build();
```

```
            cw.putMetricAlarm(request);
            System.out.printf("Created PromQL alarm %s for query %s.%n",
alarmName, query);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

Create an alarm that evaluates a single CloudWatch metric.

```
    /**
     * Creates an alarm based on the configuration provided in a JSON file.
     *
     * @param fileName the name of the JSON file containing the alarm
configuration
     * @return a CompletableFuture that represents the asynchronous operation of
creating the alarm
     * @throws RuntimeException if an exception occurs while reading the JSON
file or creating the alarm
     */
    public CompletableFuture<String> createAlarmAsync(String fileName) {
        com.fasterxml.jackson.databind.JsonNode rootNode;
        try {
            JsonParser parser = new JsonFactory().createParser(new
File(fileName));
            rootNode = new ObjectMapper().readTree(parser);
        } catch (IOException e) {
            throw new RuntimeException("Failed to read the alarm configuration
file", e);
        }

        // Extract values from the JSON node.
        String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
        String customMetricName =
rootNode.findValue("customMetricName").asText();
        String alarmName = rootNode.findValue("exampleAlarmName").asText();
        String emailTopic = rootNode.findValue("emailTopic").asText();
        String accountId = rootNode.findValue("accountId").asText();
```

```
        String region = rootNode.findValue("region").asText();

        // Create a List for alarm actions.
        List<String> alarmActions = new ArrayList<>();
        alarmActions.add("arn:aws:sns:" + region + ":" + accountId + ":" +
emailTopic);

        PutMetricAlarmRequest alarmRequest = PutMetricAlarmRequest.builder()
            .alarmActions(alarmActions)
            .alarmDescription("Example metric alarm")
            .alarmName(alarmName)

.comparisonOperator(ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD)
            .threshold(100.00)
            .metricName(customMetricName)
            .namespace(customMetricNamespace)
            .evaluationPeriods(1)
            .period(10)
            .statistic("Maximum")
            .datapointsToAlarm(1)
            .treatMissingData("ignore")
            .build();

        // Call the putMetricAlarm asynchronously and handle the result.
        return getAsyncClient().putMetricAlarm(alarmRequest)
            .handle((response, ex) -> {
                if (ex != null) {
                    logger.info("Failed to create alarm: {}", ex.getMessage());
                    throw new RuntimeException("Failed to create alarm", ex);
                } else {
                    logger.info("{} was successfully created!", alarmName);
                    return alarmName;
                }
            });
    }
```

- For API details, see PutMetricAlarm in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Create an alarm that evaluates a PromQL query over OpenTelemetry metrics.
//
// A PromQL alarm differs from a classic metric alarm in a few ways. The query
 can match
// many series at once, and each matching series is tracked separately as a
 contributor
// (see describe-alarm-contributors.js). Instead of counting breaching periods,
 you
// specify durations: a contributor moves to ALARM after it breaches continuously
 for
// PendingPeriod seconds, and back to OK after it stops breaching for
 RecoveryPeriod
// seconds. A PromQL alarm starts in OK rather than INSUFFICIENT_DATA.
//
// EvaluationCriteria is a union and is mutually exclusive with the classic
 MetricName
// and Metrics parameters. When you use it you must also set EvaluationInterval,
 and you
// must not set Period, Statistic, Threshold, ComparisonOperator,
 EvaluationPeriods,
// DatapointsToAlarm, or TreatMissingData.
const run = async () => {
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set CLOUDWATCH_ALARM_NAME to
 the name of the alarm to create.
    AlarmDescription: "Average CPU over 80% per host for the checkout service.",
    EvaluationCriteria: {
      PromQLCriteria: {
```

```
        // The comparison belongs in the query itself. There is no separate
 Threshold.
        Query:
          'avg by (host_name) (cpu_utilization_percent{service_name="checkout"})
 > 80',
        PendingPeriod: 300,
        RecoveryPeriod: 120,
      },
    },
    // How often to run the query, in seconds. Valid values are 10, 20, 30, and
 any
    // multiple of 60, up to 3600.
    EvaluationInterval: 30,
    ActionsEnabled: false,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create an alarm that evaluates a single CloudWatch metric.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
 CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
```

```
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
 the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see PutMetricAlarm in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see PutMetricAlarm in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
suspend fun putPromQlMetricAlarm(
    alarmNameVal: String,
    queryVal: String,
    evaluationIntervalVal: Int = 60,
    pendingPeriodVal: Int = 300,
    recoveryPeriodVal: Int = 120,
) {
    // The comparison belongs in the query itself. A PromQL alarm has no separate
    // threshold, comparison operator, statistic, period, or evaluation periods.
    //
    // Note that the Kotlin SDK spells this AlarmPromQlCriteria, with a lowercase
 l in
    // "Ql". Every other AWS SDK spells it PromQL, so don't be thrown by the
 difference
    // when comparing this example against the other language versions.
    val promQlCriteria =
        AlarmPromQlCriteria {
            query = queryVal
            pendingPeriod = pendingPeriodVal
            recoveryPeriod = recoveryPeriodVal
        }

    // EvaluationCriteria is a union and is mutually exclusive with the classic
    // metricName and metrics parameters. When you use it, you must also set
    // evaluationInterval.
    val request =
        PutMetricAlarmRequest {
            alarmName = alarmNameVal
            alarmDescription = "A PromQL alarm created by the Kotlin SDK"
            evaluationCriteria =
 EvaluationCriteria.PromQlCriteria(promQlCriteria)
```

```
            evaluationInterval = evaluationIntervalVal
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created PromQL alarm $alarmNameVal for query
 $queryVal")
    }
}
```

Create an alarm that evaluates a single CloudWatch metric.

```
suspend fun putMetricAlarm(
    alarmNameVal: String,
    instanceIdVal: String,
) {
    val dimensionOb =
        Dimension {
            name = "InstanceId"
            value = instanceIdVal
        }

    val request =
        PutMetricAlarmRequest {
            alarmName = alarmNameVal
            comparisonOperator = ComparisonOperator.GreaterThanThreshold
            evaluationPeriods = 1
            metricName = "CPUUtilization"
            namespace = "AWS/EC2"
            period = 60
            statistic = Statistic.fromValue("Average")
            threshold = 70.0
            actionsEnabled = false
            alarmDescription = "An Alarm created by the Kotlin SDK when server
 CPU utilization exceeds 70%"
            unit = StandardUnit.fromValue("Seconds")
            dimensions = listOf(dimensionOb)
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricAlarm(request)
        println("Successfully created an alarm with name $alarmNameVal")
```

```
    }
}
```

- For API details, see PutMetricAlarm in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def create_promql_alarm(
        self,
```

```
        alarm_name,
        query,
        evaluation_interval,
        pending_period=300,
        recovery_period=120,
        description=None,
        alarm_actions=None,
    ):
        """
        Creates an alarm that evaluates a PromQL query.

        A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
        match many series at once, and each matching series is tracked separately
as a
        *contributor*. Instead of counting breaching periods, you specify
durations: a
        contributor moves to ALARM after it breaches continuously for the pending
        period, and back to OK after it stops breaching for the recovery period.
A
        PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA.

        The PromQL evaluation parameters live in the EvaluationCriteria union,
which is
        mutually exclusive with the classic MetricName and Metrics parameters.
When you
        use EvaluationCriteria you must also set EvaluationInterval, and you must
not
        set Period, Statistic, Threshold, ComparisonOperator, EvaluationPeriods,
        DatapointsToAlarm, or TreatMissingData.

        :param alarm_name: The name of the alarm. Must be unique within the
Region.
        :param query: The PromQL query to evaluate, such as
                      'avg(cpu_utilization_percent) > 80'. The comparison belongs
in
                      the query itself; there is no separate threshold parameter.
        :param evaluation_interval: How often, in seconds, to run the query.
Valid
                                    values are 10, 20, 30, and any multiple of
60, up
                                    to 3600.
        :param pending_period: How long, in seconds, a contributor must breach
                               continuously before it moves to ALARM.
```

```
        :param recovery_period: How long, in seconds, a contributor must stop
 breaching
                                before it moves back to OK.
        :param description: The description of the alarm.
        :param alarm_actions: A list of ARNs to notify when the alarm fires, such
 as an
                              Amazon SNS topic.
        """
        promql_criteria = {
            "Query": query,
            "PendingPeriod": pending_period,
            "RecoveryPeriod": recovery_period,
        }
        kwargs = {
            "AlarmName": alarm_name,
            "EvaluationCriteria": {"PromQLCriteria": promql_criteria},
            "EvaluationInterval": evaluation_interval,
        }
        if description is not None:
            kwargs["AlarmDescription"] = description
        if alarm_actions is not None:
            kwargs["AlarmActions"] = alarm_actions

        try:
            self.cloudwatch_client.put_metric_alarm(**kwargs)
            logger.info("Created PromQL alarm %s for query %s.", alarm_name,
 query)
        except ClientError:
            logger.exception("Couldn't create PromQL alarm %s.", alarm_name)
            raise
```

Create an alarm that evaluates a single CloudWatch metric.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
   def create_metric_alarm(
       self,
       metric_namespace,
       metric_name,
       alarm_name,
       stat_type,
       period,
       eval_periods,
       threshold,
       comparison_op,
   ):
       """
       Creates an alarm that watches a metric.

       :param metric_namespace: The namespace of the metric.
       :param metric_name: The name of the metric.
       :param alarm_name: The name of the alarm.
       :param stat_type: The type of statistic the alarm watches.
       :param period: The period in which metric data are grouped to calculate
                      statistics.
       :param eval_periods: The number of periods that the metric must be over
the
                            alarm threshold before the alarm is set into an
alarmed
                            state.
       :param threshold: The threshold value to compare against the metric
statistic.
       :param comparison_op: The comparison operation used to compare the
threshold
                             against the metric.
       :return: The newly created alarm.
       """
       try:
           metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
           alarm = metric.put_alarm(
               AlarmName=alarm_name,
               Statistic=stat_type,
               Period=period,
               EvaluationPeriods=eval_periods,
               Threshold=threshold,
               ComparisonOperator=comparison_op,
```

```
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm
```

- For API details, see PutMetricAlarm in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an alarm that evaluates a PromQL query against OpenTelemetry metrics.

```
# Creates or updates an alarm that evaluates a PromQL query.
#
# A PromQL alarm differs from a classic metric alarm in a few ways. The query can
 match
# many series at once, and each matching series is tracked separately as a
 contributor.
# Instead of counting breaching periods, you specify durations: a contributor
 moves to
```

```
# ALARM after it breaches continuously for the pending period, and back to OK
 after it
# stops breaching for the recovery period. A PromQL alarm starts in the OK state
 rather
# than INSUFFICIENT_DATA.
#
# The +evaluation_criteria+ union is mutually exclusive with the classic
 +metric_name+
# and +metrics+ parameters. When you use it you must also set
 +evaluation_interval+, and
# you must not set +period+, +statistic+, +threshold+, +comparison_operator+,
# +evaluation_periods+, +datapoints_to_alarm+, or +treat_missing_data+.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the alarm, unique within the Region.
# @param criteria [Hash] The PromQL criteria, mirroring the +prom_ql_criteria+
 shape:
#   * +:query+ [String] The PromQL query to evaluate, such as
#     'avg(cpu_utilization_percent) > 80'. The comparison belongs in the query
 itself;
#     there is no separate threshold parameter.
#   * +:pending_period+ [Integer] How long, in seconds, a contributor must breach
#     continuously before it moves to ALARM.
#   * +:recovery_period+ [Integer] How long, in seconds, a contributor must stop
#     breaching before it moves back to OK.
# @param evaluation_interval [Integer] How often, in seconds, to run the query.
 Valid
#   values are 10, 20, 30, and any multiple of 60, up to 3600.
# @param alarm_description [String] A description of the alarm.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
def promql_alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  criteria,
  evaluation_interval,
  alarm_description
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    evaluation_criteria: {
      prom_ql_criteria: {
        query: criteria[:query],
```

```
        pending_period: criteria[:pending_period],
        recovery_period: criteria[:recovery_period]
      }
    },
    evaluation_interval: evaluation_interval
  )
  true
rescue StandardError => e
  puts "Error creating PromQL alarm: #{e.message}"
  false
end
```

Create an alarm that evaluates a single CloudWatch metric.

```
# Creates or updates an alarm in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param alarm_name [String] The name of the alarm.
# @param alarm_description [String] A description about the alarm.
# @param metric_name [String] The name of the metric associated with the alarm.
# @param alarm_actions [Array] A list of Strings representing the
#   Amazon Resource Names (ARNs) to execute when the alarm transitions to the
#   ALARM state.
# @param namespace [String] The namespace for the metric to alarm on.
# @param statistic [String] The statistic for the metric.
# @param dimensions [Array] A list of dimensions for the metric, specified as
#   Aws::CloudWatch::Types::Dimension.
# @param period [Integer] The number of seconds before re-evaluating the metric.
# @param unit [String] The unit of measure for the statistic.
# @param evaluation_periods [Integer] The number of periods over which data is
#   compared to the specified threshold.
# @param theshold [Float] The value against which the specified statistic is
 compared.
# @param comparison_operator [String] The arithmetic operation to use when
#   comparing the specified statistic and threshold.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
# @example
#   exit 1 unless alarm_created_or_updated?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'ObjectsInBucket',
#     'Objects exist in this bucket for more than 1 day.',
```

```
#     'NumberOfObjects',
#     ['arn:aws:sns:us-east-1:111111111111:Default_CloudWatch_Alarms_Topic'],
#     'AWS/S3',
#     'Average',
#     [
#       {
#         name: 'BucketName',
#         value: 'amzn-s3-demo-bucket'
#       },
#       {
#         name: 'StorageType',
#         value: 'AllStorageTypes'
#       }
#     ],
#     86_400,
#     'Count',
#     1,
#     1,
#     'GreaterThanThreshold'
#   )
def alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  alarm_description,
  metric_name,
  alarm_actions,
  namespace,
  statistic,
  dimensions,
  period,
  unit,
  evaluation_periods,
  threshold,
  comparison_operator
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    metric_name: metric_name,
    alarm_actions: alarm_actions,
    namespace: namespace,
    statistic: statistic,
    dimensions: dimensions,
    period: period,
```

```
    unit: unit,
    evaluation_periods: evaluation_periods,
    threshold: threshold,
    comparison_operator: comparison_operator
  )
  true
rescue StandardError => e
  puts "Error creating alarm: #{e.message}"
  false
end
```

- For API details, see PutMetricAlarm in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        lo_cwt->putmetricalarm(
          iv_alarmname                = iv_alarm_name
          iv_comparisonoperator       = iv_comparison_operator
          iv_evaluationperiods        = iv_evaluation_periods
          iv_metricname               = iv_metric_name
          iv_namespace                = iv_namespace
          iv_statistic                = iv_statistic
          iv_threshold                = iv_threshold
          iv_actionsenabled           = iv_actions_enabled
          iv_alarmdescription         = iv_alarm_description
          iv_unit                     = iv_unit
          iv_period                   = iv_period
          it_dimensions               = it_dimensions ).
        MESSAGE 'Alarm created.' TYPE 'I'.
      CATCH /aws1/cx_cwtlimitexceededfault.
        MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
```

```
    ENDTRY.
```

- For API details, see PutMetricAlarm in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutMetricData` with an AWS SDK or CLI

The following code examples show how to use `PutMetricData`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Manage custom metrics and alarms

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Wrapper to add metric data to a CloudWatch metric.
/// </summary>
/// <param name="metricNamespace">The namespace of the metric.</param>
/// <param name="metricData">A data object for the metric data.</param>
/// <returns>True if successful.</returns>
public async Task<bool> PutMetricData(string metricNamespace,
    List<MetricDatum> metricData)
{
    var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
        new PutMetricDataRequest()
        {
```

```
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutMetricData in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/PutMetricDataRequest.h>
#include <iostream>
```

Put data into the metric.

```
        Aws::CloudWatch::CloudWatchClient cw;

        Aws::CloudWatch::Model::Dimension dimension;
        dimension.SetName("UNIQUE_PAGES");
        dimension.SetValue("URLS");

        Aws::CloudWatch::Model::MetricDatum datum;
        datum.SetMetricName("PAGES_VISITED");
        datum.SetUnit(Aws::CloudWatch::Model::StandardUnit::None);
        datum.SetValue(data_point);
        datum.AddDimensions(dimension);
```

```
        Aws::CloudWatch::Model::PutMetricDataRequest request;
        request.SetNamespace("SITE/TRAFFIC");
        request.AddMetricData(datum);

        auto outcome = cw.PutMetricData(request);
        if (!outcome.IsSuccess())
        {
            std::cout << "Failed to put sample metric data:" <<
                outcome.GetError().GetMessage() << std::endl;
        }
        else
        {
            std::cout << "Successfully put sample metric data" << std::endl;
        }
```

- For API details, see PutMetricData in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To publish a custom metric to Amazon CloudWatch

The following `put-metric-data` example publishes a custom metric to Amazon CloudWatch.

```
aws cloudwatch put-metric-data \
    --namespace "Usage Metrics" \
    --metric-data file://metric.json
```

Contents of `metric.json`:

```
[
    {
        "MetricName": "New Posts",
        "Timestamp": "Wednesday, June 12, 2013 8:28:20 PM",
        "Value": 0.50,
        "Unit": "Count"
    }
]
```

This command produces no output.

For more information, see Publish custom metrics (PutMetricData / EMF) in the *Amazon CloudWatch User Guide*.

Example 2: To specify multiple dimensions

The following `put-metric-data` example illustrates how to specify multiple dimensions. Each dimension is specified as a Name=Value pair. Multiple dimensions are separated by a comma.

```
aws cloudwatch put-metric-data \
    --metric-name Buffers \
    --namespace MyNameSpace \
    --unit Bytes \
    --value 231434333 \
    --dimensions InstanceID=1-23456789,InstanceType=m1.small
```

This command produces no output.

For more information, see Metrics concepts in the *Amazon CloudWatch User Guide*.

- For API details, see PutMetricData in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Adds metric data for an alarm asynchronously.
     *
     * @param fileName the name of the JSON file containing the metric data
     * @return a CompletableFuture that asynchronously returns the
 PutMetricDataResponse
     */
```

```
   public CompletableFuture<PutMetricDataResponse>
addMetricDataForAlarmAsync(String fileName) {
       CompletableFuture<String> readFileFuture =
CompletableFuture.supplyAsync(() -> {
           try {
               JsonParser parser = new JsonFactory().createParser(new
File(fileName));
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
               return rootNode.toString(); // Return JSON as a string for
further processing
           } catch (IOException e) {
               throw new RuntimeException("Failed to read file", e);
           }
       });

       return readFileFuture.thenCompose(jsonContent -> {
           try {
               com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(jsonContent);
               String customMetricNamespace =
rootNode.findValue("customMetricNamespace").asText();
               String customMetricName =
rootNode.findValue("customMetricName").asText();
               Instant instant = Instant.now();

               // Create MetricDatum objects.
               MetricDatum datum1 = MetricDatum.builder()
                   .metricName(customMetricName)
                   .unit(StandardUnit.NONE)
                   .value(1001.00)
                   .timestamp(instant)
                   .build();

               MetricDatum datum2 = MetricDatum.builder()
                   .metricName(customMetricName)
                   .unit(StandardUnit.NONE)
                   .value(1002.00)
                   .timestamp(instant)
                   .build();

               List<MetricDatum> metricDataList = new ArrayList<>();
               metricDataList.add(datum1);
               metricDataList.add(datum2);
```

```
                // Build the PutMetricData request.
                PutMetricDataRequest request = PutMetricDataRequest.builder()
                    .namespace(customMetricNamespace)
                    .metricData(metricDataList)
                    .build();

                // Send the request asynchronously.
                return getAsyncClient().putMetricData(request);

            } catch (IOException e) {
                CompletableFuture<PutMetricDataResponse> failedFuture = new
CompletableFuture<>();
                failedFuture.completeExceptionally(new RuntimeException("Failed
to parse JSON content", e));
                return failedFuture;
            }
        }).whenComplete((response, exception) -> {
            if (exception != null) {
                logger.error("Failed to put metric data: " +
exception.getMessage(), exception);
            } else {
                logger.info("Added metric values for metric.");
            }
        });
    }
```

- For API details, see PutMetricData in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Import the SDK and client modules and call the API.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
    Namespace: "SITE/TRAFFIC",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Create the client in a separate module and export it.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see PutMetricData in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
```

```
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see PutMetricData in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun addMetricData(
    namespaceVal: String,
    metricNameVal: String,
) {
    val time =
        aws.smithy.kotlin.runtime.time.Instant
            .now()

    val datum =
        MetricDatum {
            metricName = metricNameVal
            unit = StandardUnit.None
            value = 1001.00
            timestamp = time
        }

    val datum2 =
        MetricDatum {
            metricName = metricNameVal
            unit = StandardUnit.None
            value = 1002.00
            timestamp = time
        }

    val request =
```

```
        PutMetricDataRequest {
            namespace = namespaceVal
            metricData = listOf(datum, datum2)
        }

    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.putMetricData(request)
        println("Added metric values for metric $metricNameVal.")
    }
}
```

- For API details, see PutMetricData in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell V4

Example 1: Creates a new MetricDatum object, and writes it to Amazon Web Services CloudWatch Metrics.

```
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50

### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- For API details, see PutMetricData in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Creates a new MetricDatum object, and writes it to Amazon Web Services CloudWatch Metrics.

```
### Create a MetricDatum .NET object
$Metric = New-Object -TypeName Amazon.CloudWatch.Model.MetricDatum
$Metric.Timestamp = [DateTime]::UtcNow
$Metric.MetricName = 'CPU'
$Metric.Value = 50
```

```
### Write the metric data to the CloudWatch service
Write-CWMetricData -Namespace instance1 -MetricData $Metric
```

- For API details, see PutMetricData in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
 given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
```

```
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
 name)
            raise
```

Put a set of data into a CloudWatch metric.

```
class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource


    def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
        """
        Sends a set of data to CloudWatch for a metric. All of the data in the
 set
        have the same timestamp and unit.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param timestamp: The UTC timestamp for the metric.
        :param unit: The unit of the metric.
        :param data_set: The set of data to send. This set is a dictionary that
                         contains a list of values and a list of corresponding
 counts.
                         The value and count lists must be the same length.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[
                    {
                        "MetricName": name,
```

```
                        "Timestamp": timestamp,
                        "Values": data_set["values"],
                        "Counts": data_set["counts"],
                        "Unit": unit,
                    }
                ],
            )
            logger.info("Put data set for metric %s.%s.", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
            raise
```

- For API details, see PutMetricData in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-cloudwatch'

# Adds a datapoint to a metric in Amazon CloudWatch.
#
# @param cloudwatch_client [Aws::CloudWatch::Client]
#   An initialized CloudWatch client.
# @param metric_namespace [String] The namespace of the metric to add the
#   datapoint to.
# @param metric_name [String] The name of the metric to add the datapoint to.
# @param dimension_name [String] The name of the dimension to add the
#   datapoint to.
# @param dimension_value [String] The value of the dimension to add the
#   datapoint to.
# @param metric_value [Float] The value of the datapoint.
# @param metric_unit [String] The unit of measurement for the datapoint.
```

```
# @return [Boolean]
# @example
#   exit 1 unless datapoint_added_to_metric?(
#     Aws::CloudWatch::Client.new(region: 'us-east-1'),
#     'SITE/TRAFFIC',
#     'UniqueVisitors',
#     'SiteName',
#     'example.com',
#     5_885.0,
#     'Count'
#   )
def datapoint_added_to_metric?(
  cloudwatch_client,
  metric_namespace,
  metric_name,
  dimension_name,
  dimension_value,
  metric_value,
  metric_unit
)
  cloudwatch_client.put_metric_data(
    namespace: metric_namespace,
    metric_data: [
      {
        metric_name: metric_name,
        dimensions: [
          {
            name: dimension_name,
            value: dimension_value
          }
        ],
        value: metric_value,
        unit: metric_unit
      }
    ]
  )
  puts "Added data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}'."
  true
rescue StandardError => e
  puts "Error adding data about '#{metric_name}' to namespace " \
    "'#{metric_namespace}': #{e.message}"
  false
end
```

- For API details, see PutMetricData in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
DATA lt_metricdata TYPE /aws1/cl_cwtmetricdatum=>tt_metricdata.

"Create metric data object.
DATA(lo_metricdatum) = NEW /aws1/cl_cwtmetricdatum(
  iv_metricname = iv_metric_name
  iv_value      = iv_value
  iv_unit       = iv_unit ).

INSERT lo_metricdatum INTO TABLE lt_metricdata.

TRY.
    lo_cwt->putmetricdata(
      iv_namespace   = iv_namespace
      it_metricdata  = lt_metricdata ).
    MESSAGE 'Metric data added.' TYPE 'I'.
  CATCH /aws1/cx_cwtinvparamvalueex.
    MESSAGE 'The specified argument was not valid.' TYPE 'E'.
ENDTRY.
```

Put a set of data into a CloudWatch metric.

```
DATA lt_metricdata TYPE /aws1/cl_cwtmetricdatum=>tt_metricdata.
```

```
    "Create metric data object with values and counts.
    DATA(lo_metricdatum) = NEW /aws1/cl_cwtmetricdatum(
      iv_metricname = iv_metric_name
      iv_timestamp  = iv_timestamp
      iv_unit       = iv_unit
      it_values     = it_values
      it_counts     = it_counts ).

    INSERT lo_metricdatum INTO TABLE lt_metricdata.

    TRY.
        lo_cwt->putmetricdata(
          iv_namespace   = iv_namespace
          it_metricdata  = lt_metricdata ).
        MESSAGE 'Metric data set added.' TYPE 'I'.
      CATCH /aws1/cx_cwtinvparamvalueex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
    ENDTRY.
```

- For API details, see PutMetricData in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `StartOTelEnrichment` with an AWS SDK

The following code examples show how to use `StartOTelEnrichment`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Turn on OTel enrichment for the account. Once enrichment is running,
CloudWatch
    /// vended metrics that carry a resource identifier dimension, such as the
Amazon EC2
    /// CPUUtilization metric with its InstanceId dimension, are decorated with
resource
    /// ARN and resource tag labels and become queryable with PromQL.
    ///
    /// Resource tags on telemetry must already be enabled for the account before
you
    /// call this operation.
    /// </summary>
    /// <returns>True if successful.</returns>
    public async Task<bool> StartOTelEnrichment()
    {
        var response = await _amazonCloudWatch.StartOTelEnrichmentAsync(
            new StartOTelEnrichmentRequest());

        _logger.LogInformation("Started OTel enrichment for this account.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see StartOTelEnrichment in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/StartOTelEnrichmentRequest.h>
#include <iostream>
```

Start OpenTelemetry enrichment.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::StartOTelEnrichmentRequest request;

        auto outcome = cw.StartOTelEnrichment(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to start OTel enrichment: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            std::cout << "Successfully started OTel enrichment for this account."
                      << std::endl;
        }
```

- For API details, see StartOTelEnrichment in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Turns on OTel enrichment for the account. Once enrichment is running,
CloudWatch
    * vended metrics that carry a resource identifier dimension, such as the EC2
    * CPUUtilization metric with its InstanceId dimension, are decorated with
resource
    * ARN and resource tag labels and become queryable with PromQL.
    *
    * <p>Resource tags on telemetry must already be enabled for the account
before you
    * call this operation.
    *
    * @param cw the CloudWatch client
    */
   public static void startOTelEnrichment(CloudWatchClient cw) {
       try {
           cw.startOTelEnrichment(StartOTelEnrichmentRequest.builder().build());
           System.out.println("Started OTel enrichment for this account.");

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see StartOTelEnrichment in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { StartOTelEnrichmentCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Turn on OTel enrichment for the account. Once enrichment is running,
 CloudWatch
// vended metrics that carry a resource identifier dimension - for example the
 EC2
// CPUUtilization metric with its InstanceId dimension - are decorated with
 resource
// ARN and resource tag labels, and become queryable with PromQL.
//
// Resource tags on telemetry must already be enabled for the account before you
 call
// this operation.
const run = async () => {
  const command = new StartOTelEnrichmentCommand({});

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see StartOTelEnrichment in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun startOTelEnrichment() {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.startOTelEnrichment(StartOTelEnrichmentRequest {})
        println("Successfully started OTel enrichment for this account")
    }
}
```

- For API details, see StartOTelEnrichment in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
```

```
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def start_otel_enrichment(self):
        """
        Turns on OTel enrichment for the account. Once enrichment is running,
        CloudWatch vended metrics that carry a resource identifier dimension,
such as
        the EC2 CPUUtilization metric with its InstanceId dimension, are
decorated with
        resource ARN and resource tag labels and become queryable with PromQL.

        Resource tags on telemetry must already be enabled for the account before
you
        call this operation.
        """
        try:
            # Boto3 splits the OTel prefix when it converts the
StartOTelEnrichment
            # operation name to snake case, so the method is
start_o_tel_enrichment.
            self.cloudwatch_client.start_o_tel_enrichment()
            logger.info("Started OTel enrichment for this account.")
        except ClientError:
            logger.exception("Couldn't start OTel enrichment.")
            raise
```

- For API details, see StartOTelEnrichment in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Turns on OTel enrichment for the account. Once enrichment is running,
 CloudWatch vended
# metrics that carry a resource identifier dimension, such as the Amazon EC2
# CPUUtilization metric with its InstanceId dimension, are decorated with
 resource ARN
# and resource tag labels and become queryable with PromQL.
#
# Resource tags on telemetry must already be enabled for the account before you
 call
# this operation.
#
# Note that the Ruby SDK renders the OTel prefix as +o_tel+, so the method is
# +start_o_tel_enrichment+ rather than +start_otel_enrichment+.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Boolean] true if enrichment was started; otherwise, false.
def otel_enrichment_started?(cloudwatch_client)
  cloudwatch_client.start_o_tel_enrichment
  true
rescue StandardError => e
  puts "Error starting OTel enrichment: #{e.message}"
  false
end
```

- For API details, see StartOTelEnrichment in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `StopOTelEnrichment` with an AWS SDK

The following code examples show how to use `StopOTelEnrichment`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Learn the basics
- Send OpenTelemetry metrics and alarm on them with PromQL

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Turn off OTel enrichment for the account. Existing PromQL alarms are not
deleted,
    /// but vended metrics stop being enriched with resource ARN and tag labels,
so
    /// queries that select on those labels stop matching.
    /// </summary>
    /// <returns>True if successful.</returns>
    public async Task<bool> StopOTelEnrichment()
    {
        var response = await _amazonCloudWatch.StopOTelEnrichmentAsync(
            new StopOTelEnrichmentRequest());

        _logger.LogInformation("Stopped OTel enrichment for this account.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see StopOTelEnrichment in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Include the required files.

```
#include <aws/core/Aws.h>
#include <aws/monitoring/CloudWatchClient.h>
#include <aws/monitoring/model/StopOTelEnrichmentRequest.h>
#include <iostream>
```

Stop OpenTelemetry enrichment.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
        Aws::CloudWatch::CloudWatchClient cw(clientConfig);

        Aws::CloudWatch::Model::StopOTelEnrichmentRequest request;

        auto outcome = cw.StopOTelEnrichment(request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to stop OTel enrichment: "
                      << outcome.GetError().GetMessage() << std::endl;
        } else {
            std::cout << "Successfully stopped OTel enrichment for this account."
                      << std::endl;
        }
```

- For API details, see StopOTelEnrichment in *AWS SDK for C++ API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Turns off OTel enrichment for the account. Existing PromQL alarms are not
     * deleted, but vended metrics stop being enriched with resource ARN and tag
labels,
     * so queries that select on those labels stop matching.
     *
     * @param cw the CloudWatch client
     */
    public static void stopOTelEnrichment(CloudWatchClient cw) {
        try {
            cw.stopOTelEnrichment(StopOTelEnrichmentRequest.builder().build());
            System.out.println("Stopped OTel enrichment for this account.");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see StopOTelEnrichment in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import { StopOTelEnrichmentCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

// Turn off OTel enrichment for the account. Existing PromQL alarms are not
 deleted,
// but vended metrics stop being enriched with resource ARN and tag labels, so
 queries
// that select on those labels stop matching.
const run = async () => {
  const command = new StopOTelEnrichmentCommand({});

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- For API details, see StopOTelEnrichment in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun stopOTelEnrichment() {
    CloudWatchClient.fromEnvironment { region = "us-east-1" }.use { cwClient ->
        cwClient.stopOTelEnrichment(StopOTelEnrichmentRequest {})
        println("Successfully stopped OTel enrichment for this account")
    }
}
```

- For API details, see StopOTelEnrichment in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
```

```
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))


    def stop_otel_enrichment(self):
        """
        Turns off OTel enrichment for the account. Existing PromQL alarms are not
        deleted, but vended metrics stop being enriched with resource ARN and tag
        labels, so queries that select on those labels stop matching.
        """
        try:
            self.cloudwatch_client.stop_o_tel_enrichment()
            logger.info("Stopped OTel enrichment for this account.")
        except ClientError:
            logger.exception("Couldn't stop OTel enrichment.")
            raise
```

- For API details, see StopOTelEnrichment in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# Turns off OTel enrichment for the account. Existing PromQL alarms are not
 deleted, but
```

```
# vended metrics stop being enriched with resource ARN and tag labels, so queries
 that
# select on those labels stop matching.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Boolean] true if enrichment was stopped; otherwise, false.
def otel_enrichment_stopped?(cloudwatch_client)
  cloudwatch_client.stop_o_tel_enrichment
  true
rescue StandardError => e
  puts "Error stopping OTel enrichment: #{e.message}"
  false
end
```

- For API details, see StopOTelEnrichment in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudWatch using AWS SDKs

The following code examples show you how to implement common scenarios in CloudWatch with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudWatch or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Creating a monitoring dashboard with function name as a variable
- Get started with CloudWatch alarms using an AWS SDK
- Manage CloudWatch custom metrics and alarms using an AWS SDK
- Monitor performance of Amazon DynamoDB using an AWS SDK
- Run CPU stress tests on virtual machine instances using fault injection

- Send OpenTelemetry metrics to CloudWatch and alarm on them with PromQL using an AWS SDK
- Using property variables in monitoring dashboards to monitor multiple serverless functions

Creating a monitoring dashboard with function name as a variable

The following code example shows how to:

- Create a CloudWatch dashboard
- Add Lambda metrics widgets with a function name variable
- Verify the dashboard
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Script to create a CloudWatch dashboard with Lambda function name as a variable
# This script creates a CloudWatch dashboard that allows you to switch between
 different Lambda functions

# Set up logging with secure permissions
LOG_FILE="${HOME}/.cloudwatch-dashboard-script.log"
touch "$LOG_FILE" && chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "$(date): Starting CloudWatch dashboard creation script"

# Security: Set strict error handling
set -uo pipefail
trap 'handle_error "Script failed at line $LINENO"' ERR
```

```
# Function to handle errors
handle_error() {
    local error_msg="${1:-Unknown error}"
    echo "ERROR: $error_msg" >&2
    echo "Resources created:"
    echo "- CloudWatch Dashboard: LambdaMetricsDashboard"
    echo ""
    echo "=========================================="
    echo "CLEANUP CONFIRMATION"
    echo "=========================================="
    echo "An error occurred. Proceeding with automatic cleanup..."

    echo "Cleaning up resources..."
    aws cloudwatch delete-dashboards --dashboard-names LambdaMetricsDashboard 2>/
dev/null || true

    # Clean up temporary files
    if [ -n "${TEMP_DIR:-}" ] && [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi
    rm -f dashboard-body.json

    echo "Cleanup complete."
    exit 1
}

# Security: Validate AWS CLI is installed
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed. Please install it and try again."
fi

# Check if AWS CLI is installed and configured
echo "Checking AWS CLI configuration..."
if ! aws sts get-caller-identity > /dev/null 2>&1; then
    handle_error "AWS CLI is not properly configured. Please configure it with
 'aws configure' and try again."
fi

# Get the current region securely
REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $REGION"
fi
```

```
echo "Using region: $REGION"

# Validate region format
if ! [[ "$REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
    handle_error "Invalid AWS region format: $REGION"
fi

# Check if there are any Lambda functions in the account
echo "Checking for Lambda functions..."
LAMBDA_FUNCTIONS=$(aws lambda list-functions --region "$REGION" --query
 "Functions[*].FunctionName" --output text 2>/dev/null || echo "")

if [ -z "$LAMBDA_FUNCTIONS" ]; then
    echo "No Lambda functions found in your account. Creating a simple test
 function..."

    # Create a temporary directory for Lambda function code with secure
 permissions
    TEMP_DIR=$(mktemp -d)
    chmod 700 "$TEMP_DIR"
    trap 'rm -rf "$TEMP_DIR"' EXIT

    # Create a simple Lambda function
    cat > "$TEMP_DIR/index.js" << 'EOF'
exports.handler = async (event) => {
    console.log('Event:', JSON.stringify(event, null, 2));
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
};
EOF

    # Zip the function code
    if ! cd "$TEMP_DIR"; then
        handle_error "Failed to change to temporary directory"
    fi

    if ! zip -q function.zip index.js; then
        handle_error "Failed to create zip file"
    fi

    # Create a role for the Lambda function with restricted trust policy
    ROLE_NAME="LambdaDashboardTestRole-$(date +%s)"
```

```
    TRUST_POLICY='{"Version":"2012-10-17",       "Statement":
[{"Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}'

    if ! ROLE_ARN=$(aws iam create-role \
        --role-name "$ROLE_NAME" \
        --assume-role-policy-document "$TRUST_POLICY" \
        --tags Key=project,Value=doc-smith Key=tutorial,Value=cloudwatch-
dynamicdash \
        --query "Role.Arn" \
        --output text 2>/dev/null); then
        handle_error "Failed to create IAM role for Lambda function"
    fi

    echo "Waiting for role to be available..."
    sleep 10

    # Attach basic Lambda execution policy
    if ! aws iam attach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"; then
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to attach policy to IAM role"
    fi

    # Create the Lambda function
    FUNCTION_NAME="DashboardTestFunction-$(date +%s)"
    if ! aws lambda create-function \
        --function-name "$FUNCTION_NAME" \
        --runtime nodejs18.x \
        --role "$ROLE_ARN" \
        --handler index.handler \
        --zip-file fileb://function.zip \
        --tags project=doc-smith,tutorial=cloudwatch-dynamicdash \
        --region "$REGION" > /dev/null 2>&1; then
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to create Lambda function"
    fi
```

```
    # Invoke the function to generate some metrics
    echo "Invoking Lambda function to generate metrics..."
    for i in {1..5}; do
        aws lambda invoke --function-name "$FUNCTION_NAME" --payload '{}' /dev/
null --region "$REGION" > /dev/null 2>&1 || true
        sleep 1
    done

    # Go back to original directory
    cd - > /dev/null

    # Set the function name for the dashboard
    DEFAULT_FUNCTION="$FUNCTION_NAME"
else
    # Use the first Lambda function as default
    DEFAULT_FUNCTION=$(echo "$LAMBDA_FUNCTIONS" | awk '{print $1}')
    echo "Found Lambda functions. Using $DEFAULT_FUNCTION as default."
    FUNCTION_NAME=""
    ROLE_NAME=""
fi

# Create a dashboard with Lambda metrics and a function name variable
echo "Creating CloudWatch dashboard with Lambda function name variable..."

# Create a JSON file for the dashboard body with secure permissions
DASHBOARD_JSON="dashboard-body-$$.json"
touch "$DASHBOARD_JSON" && chmod 600 "$DASHBOARD_JSON"

# Escape special characters in region and function name for JSON
REGION_ESCAPED=$(printf '%s\n' "$REGION" | sed 's:[\/&]:\\&:g')
FUNCTION_ESCAPED=$(printf '%s\n' "$DEFAULT_FUNCTION" | sed 's:[\/&]:\\&:g')

cat > "$DASHBOARD_JSON" << EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "\${FunctionName}" ],
```

```
          [ ".", "Errors", ".", "." ],
          [ ".", "Throttles", ".", "." ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Lambda Function Metrics for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 0,
      "y": 6,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Duration", "FunctionName", "\${FunctionName}",
 { "stat": "Average" } ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Duration for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 12,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "ConcurrentExecutions", "FunctionName", "\
${FunctionName}" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Concurrent Executions for \${FunctionName}",
        "period": 300
```

```
      }
    }
  ],
  "periodOverride": "auto",
  "variables": [
    {
      "type": "property",
      "id": "FunctionName",
      "property": "FunctionName",
      "label": "Lambda Function",
      "inputType": "select",
      "values": [
        {
          "value": "$FUNCTION_ESCAPED",
          "label": "$FUNCTION_ESCAPED"
        }
      ]
    }
  ]
}
EOF

# Validate JSON before sending
if ! jq empty "$DASHBOARD_JSON" 2>/dev/null; then
    handle_error "Invalid JSON generated for dashboard"
fi

# Create the dashboard using the JSON file
DASHBOARD_NAME="LambdaMetricsDashboard-$(date +%s)"
if ! DASHBOARD_RESULT=$(aws cloudwatch put-dashboard \
    --dashboard-name "$DASHBOARD_NAME" \
    --dashboard-body file://"$DASHBOARD_JSON" \
    --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to create CloudWatch dashboard."
```

```
fi

# Display any validation messages but continue
if echo "$DASHBOARD_RESULT" | grep -q "DashboardValidationMessages"; then
    echo "Dashboard created with validation messages:"
    echo "$DASHBOARD_RESULT"
    echo "These validation messages are warnings and the dashboard should still
 function."
else
    echo "Dashboard created successfully!"
fi

# Verify the dashboard was created
echo "Verifying dashboard creation..."
if ! DASHBOARD_INFO=$(aws cloudwatch get-dashboard --dashboard-name
 "$DASHBOARD_NAME" --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to verify dashboard creation."
fi

echo "Dashboard verification successful!"
echo "Dashboard details:"
echo "$DASHBOARD_INFO"

# List all dashboards to confirm
echo "Listing all dashboards:"
if ! DASHBOARDS=$(aws cloudwatch list-dashboards --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
```

```
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to list dashboards."
fi
echo "$DASHBOARDS"

# Show instructions for accessing the dashboard
echo ""
echo "Dashboard created successfully! To access it:"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/"
echo "2. In the navigation pane, choose Dashboards"
echo "3. Select $DASHBOARD_NAME"
echo "4. You should see a dropdown menu labeled 'Lambda Function' at the top of
 the dashboard"
echo "5. Use this dropdown to select different Lambda functions and see their
 metrics"
echo ""

# Create a list of resources for cleanup
RESOURCES=("- CloudWatch Dashboard: $DASHBOARD_NAME")
if [ -n "${FUNCTION_NAME:-}" ]; then
    RESOURCES+=("- Lambda Function: $FUNCTION_NAME")
    RESOURCES+=("- IAM Role: $ROLE_NAME")
fi

# Prompt for cleanup with automatic yes
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Resources created:"
for resource in "${RESOURCES[@]}"; do
    echo "$resource"
done
echo ""
echo "Proceeding with automatic cleanup..."

CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

    # Delete the dashboard
```

```
    if aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" --
region "$REGION" 2>/dev/null; then
        echo "Dashboard deleted successfully."
    else
        echo "WARNING: Failed to delete dashboard. You may need to delete it
 manually."
    fi

    # If we created a Lambda function, delete it and its role
    if [ -n "${FUNCTION_NAME:-}" ]; then
        echo "Deleting Lambda function..."
        if aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null; then
            echo "Lambda function deleted successfully."
        else
            echo "WARNING: Failed to delete Lambda function. You may need to
 delete it manually."
        fi

        echo "Detaching role policy..."
        if aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null; then
            echo "Role policy detached successfully."
        else
            echo "WARNING: Failed to detach role policy. You may need to detach
 it manually."
        fi

        echo "Deleting IAM role..."
        if aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null; then
            echo "IAM role deleted successfully."
        else
            echo "WARNING: Failed to delete IAM role. You may need to delete it
 manually."
        fi
    fi

    # Clean up the JSON file
    rm -f "$DASHBOARD_JSON"

    echo "Cleanup complete."
fi
```

```
echo "Script completed successfully!"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards
  - DeleteFunction
  - DeleteRole
  - DetachRolePolicy
  - GetCallerIdentity
  - GetDashboard
  - Invoke
  - ListDashboards
  - ListFunctions
  - PutDashboard

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started with CloudWatch alarms using an AWS SDK

The following code example shows how to:

- Create an alarm.
- Disable alarm actions.
- Describe an alarm.
- Delete an alarm.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
DATA lt_alarmnames TYPE /aws1/cl_cwtalarmnames_w=>tt_alarmnames.
DATA lo_alarmname TYPE REF TO /aws1/cl_cwtalarmnames_w.

"Create an alarm"
TRY.
    lo_cwt->putmetricalarm(
      iv_alarmname                = iv_alarm_name
      iv_comparisonoperator       = iv_comparison_operator
      iv_evaluationperiods        = iv_evaluation_periods
      iv_metricname               = iv_metric_name
      iv_namespace                = iv_namespace
      iv_statistic                = iv_statistic
      iv_threshold                = iv_threshold
      iv_actionsenabled           = iv_actions_enabled
      iv_alarmdescription         = iv_alarm_description
      iv_unit                     = iv_unit
      iv_period                   = iv_period
      it_dimensions               = it_dimensions ).
    MESSAGE 'Alarm created' TYPE 'I'.
  CATCH /aws1/cx_cwtlimitexceededfault.
    MESSAGE 'The request processing has exceeded the limit' TYPE 'E'.
ENDTRY.

"Create an ABAP internal table for the created alarm."
lo_alarmname = NEW #( iv_value = iv_alarm_name ).
INSERT lo_alarmname INTO TABLE lt_alarmnames.

"Disable alarm actions."
TRY.
    lo_cwt->disablealarmactions(
      it_alarmnames               = lt_alarmnames ).
```

```
        MESSAGE 'Alarm actions disabled' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_disablealarm_exception).
        DATA(lv_disablealarm_error) = |"{ lo_disablealarm_exception-
>av_err_code }" - { lo_disablealarm_exception->av_err_msg }|.
        MESSAGE lv_disablealarm_error TYPE 'E'.
    ENDTRY.

    "Describe alarm using the same ABAP internal table."
    TRY.
        oo_result = lo_cwt->describealarms(                    " oo_result is
 returned for testing purpose "
          it_alarmnames               = lt_alarmnames ).
        MESSAGE 'Alarms retrieved' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_describealarms_exception).
        DATA(lv_describealarms_error) = |"{ lo_describealarms_exception-
>av_err_code }" - { lo_describealarms_exception->av_err_msg }|.
        MESSAGE lv_describealarms_error TYPE 'E'.
    ENDTRY.

    "Delete alarm."
    TRY.
        lo_cwt->deletealarms(
          it_alarmnames = lt_alarmnames ).
        MESSAGE 'Alarms deleted' TYPE 'I'.
      CATCH /aws1/cx_cwtresourcenotfound.
        MESSAGE 'Resource being access is not found.' TYPE 'E'.
    ENDTRY.
```

- For API details, see the following topics in *AWS SDK for SAP ABAP API reference*.
  - DeleteAlarms
  - DescribeAlarms
  - DisableAlarmActions
  - PutMetricAlarm

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Manage CloudWatch custom metrics and alarms using an AWS SDK

The following code example shows how to:

- Create an alarm to watch a single CloudWatch metric.
- Put data into the metric with `PutMetricData` and trigger the alarm.
- Get data from the alarm.
- Delete the alarm.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a class that wraps CloudWatch operations.

```
from datetime import datetime, timedelta, timezone
import logging
from pprint import pprint
import random
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchWrapper:
    """Encapsulates Amazon CloudWatch functions."""

    def __init__(self, cloudwatch_resource):
        """
        :param cloudwatch_resource: A Boto3 CloudWatch resource.
        """
        self.cloudwatch_resource = cloudwatch_resource
```

```
   def put_metric_data_set(self, namespace, name, timestamp, unit, data_set):
       """
       Sends a set of data to CloudWatch for a metric. All of the data in the
set
       have the same timestamp and unit.

       :param namespace: The namespace of the metric.
       :param name: The name of the metric.
       :param timestamp: The UTC timestamp for the metric.
       :param unit: The unit of the metric.
       :param data_set: The set of data to send. This set is a dictionary that
                        contains a list of values and a list of corresponding
counts.
                        The value and count lists must be the same length.
       """
       try:
           metric = self.cloudwatch_resource.Metric(namespace, name)
           metric.put_data(
               Namespace=namespace,
               MetricData=[
                   {
                       "MetricName": name,
                       "Timestamp": timestamp,
                       "Values": data_set["values"],
                       "Counts": data_set["counts"],
                       "Unit": unit,
                   }
               ],
           )
           logger.info("Put data set for metric %s.%s.", namespace, name)
       except ClientError:
           logger.exception("Couldn't put data set for metric %s.%s.",
namespace, name)
           raise


   def create_metric_alarm(
       self,
       metric_namespace,
       metric_name,
       alarm_name,
       stat_type,
```

```
        period,
        eval_periods,
        threshold,
        comparison_op,
    ):
        """
        Creates an alarm that watches a metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :param alarm_name: The name of the alarm.
        :param stat_type: The type of statistic the alarm watches.
        :param period: The period in which metric data are grouped to calculate
                       statistics.
        :param eval_periods: The number of periods that the metric must be over
the
                             alarm threshold before the alarm is set into an
alarmed
                             state.
        :param threshold: The threshold value to compare against the metric
statistic.
        :param comparison_op: The comparison operation used to compare the
threshold
                              against the metric.
        :return: The newly created alarm.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
metric_name)
            alarm = metric.put_alarm(
                AlarmName=alarm_name,
                Statistic=stat_type,
                Period=period,
                EvaluationPeriods=eval_periods,
                Threshold=threshold,
                ComparisonOperator=comparison_op,
            )
            logger.info(
                "Added alarm %s to track metric %s.%s.",
                alarm_name,
                metric_namespace,
                metric_name,
            )
        except ClientError:
```

```
            logger.exception(
                "Couldn't add alarm %s to metric %s.%s",
                alarm_name,
                metric_namespace,
                metric_name,
            )
            raise
        else:
            return alarm


    def put_metric_data(self, namespace, name, value, unit):
        """
        Sends a single data value to CloudWatch for a metric. This metric is
given
        a timestamp of the current UTC time.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
        :param value: The value of the metric.
        :param unit: The unit of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            metric.put_data(
                Namespace=namespace,
                MetricData=[{"MetricName": name, "Value": value, "Unit": unit}],
            )
            logger.info("Put data for metric %s.%s", namespace, name)
        except ClientError:
            logger.exception("Couldn't put data for metric %s.%s", namespace,
name)
            raise


    def get_metric_statistics(self, namespace, name, start, end, period,
stat_types):
        """
        Gets statistics for a metric within a specified time span. Metrics are
grouped
        into the specified period.

        :param namespace: The namespace of the metric.
        :param name: The name of the metric.
```

```
        :param start: The UTC start time of the time span to retrieve.
        :param end: The UTC end time of the time span to retrieve.
        :param period: The period, in seconds, in which to group metrics. The
period
                       must match the granularity of the metric, which depends on
                       the metric's age. For example, metrics that are older than
                       three hours have a one-minute granularity, so the period
must
                       be at least 60 and must be a multiple of 60.
        :param stat_types: The type of statistics to retrieve, such as average
value
                           or maximum value.
        :return: The retrieved statistics for the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(namespace, name)
            stats = metric.get_statistics(
                StartTime=start, EndTime=end, Period=period,
Statistics=stat_types
            )
            logger.info(
                "Got %s statistics for %s.", len(stats["Datapoints"]),
stats["Label"]
            )
        except ClientError:
            logger.exception("Couldn't get statistics for %s.%s.", namespace,
name)
            raise
        else:
            return stats


    def get_metric_alarms(self, metric_namespace, metric_name):
        """
        Gets the alarms that are currently watching the specified metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        :returns: An iterator that yields the alarms.
        """
        metric = self.cloudwatch_resource.Metric(metric_namespace, metric_name)
        alarm_iter = metric.alarms.all()
        logger.info("Got alarms for metric %s.%s.", metric_namespace,
metric_name)
```

```
        return alarm_iter


    def delete_metric_alarms(self, metric_namespace, metric_name):
        """
        Deletes all of the alarms that are currently watching the specified
 metric.

        :param metric_namespace: The namespace of the metric.
        :param metric_name: The name of the metric.
        """
        try:
            metric = self.cloudwatch_resource.Metric(metric_namespace,
 metric_name)
            metric.alarms.delete()
            logger.info(
                "Deleted alarms for metric %s.%s.", metric_namespace, metric_name
            )
        except ClientError:
            logger.exception(
                "Couldn't delete alarms for metric %s.%s.",
                metric_namespace,
                metric_name,
            )
            raise
```

Use the wrapper class to put data in a metric, trigger an alarm that watches the metric, and get data from the alarm.

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch metrics and alarms demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    cw_wrapper = CloudWatchWrapper(boto3.resource("cloudwatch"))

    minutes = 20
    metric_namespace = "doc-example-metric"
```

```
    metric_name = "page_views"
    start = datetime.now(timezone.utc) - timedelta(minutes=minutes)
    print(
        f"Putting data into metric {metric_namespace}.{metric_name} spanning the
"
        f"last {minutes} minutes."
    )
    for offset in range(0, minutes):
        stamp = start + timedelta(minutes=offset)
        cw_wrapper.put_metric_data_set(
            metric_namespace,
            metric_name,
            stamp,
            "Count",
            {
                "values": [
                    random.randint(bound, bound * 2)
                    for bound in range(offset + 1, offset + 11)
                ],
                "counts": [random.randint(1, offset + 1) for _ in range(10)],
            },
        )

    alarm_name = "high_page_views"
    period = 60
    eval_periods = 2
    print(f"Creating alarm {alarm_name} for metric {metric_name}.")
    alarm = cw_wrapper.create_metric_alarm(
        metric_namespace,
        metric_name,
        alarm_name,
        "Maximum",
        period,
        eval_periods,
        100,
        "GreaterThanThreshold",
    )
    print(f"Alarm ARN is {alarm.alarm_arn}.")
    print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Sending data to trigger the alarm. This requires data over the
threshold "
        f"for {eval_periods} periods of {period} seconds each."
```

```
    )
    while alarm.state_value == "INSUFFICIENT_DATA":
        print("Sending data for the metric.")
        cw_wrapper.put_metric_data(
            metric_namespace, metric_name, random.randint(100, 200), "Count"
        )
        alarm.load()
        print(f"Current alarm state is: {alarm.state_value}.")
        if alarm.state_value == "INSUFFICIENT_DATA":
            print(f"Waiting for {period} seconds...")
            time.sleep(period)
        else:
            print("Wait for a minute for eventual consistency of metric data.")
            time.sleep(period)
            if alarm.state_value == "OK":
                alarm.load()
                print(f"Current alarm state is: {alarm.state_value}.")

    print(
        f"Getting data for metric {metric_namespace}.{metric_name} during
timespan "
        f"of {start} to {datetime.now(timezone.utc)} (times are UTC)."
    )
    stats = cw_wrapper.get_metric_statistics(
        metric_namespace,
        metric_name,
        start,
        datetime.now(timezone.utc),
        60,
        ["Average", "Minimum", "Maximum"],
    )
    print(
        f"Got {len(stats['Datapoints'])} data points for metric "
        f"{metric_namespace}.{metric_name}."
    )
    pprint(sorted(stats["Datapoints"], key=lambda x: x["Timestamp"]))

    print(f"Getting alarms for metric {metric_name}.")
    alarms = cw_wrapper.get_metric_alarms(metric_namespace, metric_name)
    for alarm in alarms:
        print(f"Alarm {alarm.name} is currently in state {alarm.state_value}.")

    print(f"Deleting alarms for metric {metric_name}.")
    cw_wrapper.delete_metric_alarms(metric_namespace, metric_name)
```

```
    print("Thanks for watching!")
    print("-" * 88)
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - DeleteAlarms
  - DescribeAlarmsForMetric
  - DisableAlarmActions
  - EnableAlarmActions
  - GetMetricStatistics
  - ListMetrics
  - PutMetricAlarm
  - PutMetricData

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Monitor performance of Amazon DynamoDB using an AWS SDK

The following code example shows how to configure an application's use of DynamoDB to monitor performance.

Java

SDK for Java 2.x

This example shows how to configure a Java application to monitor the performance of DynamoDB. The application sends metric data to CloudWatch where you can monitor the performance.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- CloudWatch
- DynamoDB

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Run CPU stress tests on virtual machine instances using fault injection

The following code example shows how to:

- Create IAM roles
- Create a CloudWatch alarm
- Create an experiment template
- Run the experiment
- Verify the results
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS FIS CPU Stress Test Tutorial Script
# This script automates the steps in the AWS FIS CPU stress test tutorial

#    approach using epoch time calculations that work across all Linux
 distributions
```

```
# Set up logging
LOG_FILE="fis-tutorial-$(date +%Y%m%d-%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting AWS FIS CPU Stress Test Tutorial Script"
echo "Logging to $LOG_FILE"
echo "============================================="

# Function to check for errors in command output
check_error() {
    local output=$1
    local cmd=$2

    if echo "$output" | grep -i "error" > /dev/null; then
        # Ignore specific expected errors
        if [[ "$cmd" == *"aws fis get-experiment"* ]] && [[ "$output" ==
 *"ConfigurationFailure"* ]]; then
            echo "Note: Experiment failed due to configuration issue. This is
 expected in some cases."
            return 0
        fi

        echo "ERROR: Command failed: $cmd"
        echo "Output: $output"
        cleanup_on_error
        exit 1
    fi
}

# Function to clean up resources on error
cleanup_on_error() {
    echo "Error encountered. Cleaning up resources..."

    if [ -n "$EXPERIMENT_ID" ]; then
        echo "Stopping experiment $EXPERIMENT_ID if running..."
        aws fis stop-experiment --id "$EXPERIMENT_ID" 2>/dev/null || true
    fi

    if [ -n "$TEMPLATE_ID" ]; then
        echo "Deleting experiment template $TEMPLATE_ID..."
        aws fis delete-experiment-template --id "$TEMPLATE_ID" || true
    fi

    if [ -n "$INSTANCE_ID" ]; then
```

```
        echo "Terminating EC2 instance $INSTANCE_ID..."
        aws ec2 terminate-instances --instance-ids "$INSTANCE_ID" || true
    fi

    if [ -n "$ALARM_NAME" ]; then
        echo "Deleting CloudWatch alarm $ALARM_NAME..."
        aws cloudwatch delete-alarms --alarm-names "$ALARM_NAME" || true
    fi

    if [ -n "$INSTANCE_PROFILE_NAME" ]; then
        echo "Removing role from instance profile..."
        aws iam remove-role-from-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" --role-name "$EC2_ROLE_NAME" || true

        echo "Deleting instance profile..."
        aws iam delete-instance-profile --instance-profile-name
 "$INSTANCE_PROFILE_NAME" || true
    fi

    if [ -n "$FIS_ROLE_NAME" ]; then
        echo "Deleting FIS role policy..."
        aws iam delete-role-policy --role-name "$FIS_ROLE_NAME" --policy-name
 "$FIS_POLICY_NAME" || true

        echo "Deleting FIS role..."
        aws iam delete-role --role-name "$FIS_ROLE_NAME" || true
    fi

    if [ -n "$EC2_ROLE_NAME" ]; then
        echo "Detaching policy from EC2 role..."
        aws iam detach-role-policy --role-name "$EC2_ROLE_NAME" --policy-arn
 "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore" || true

        echo "Deleting EC2 role..."
        aws iam delete-role --role-name "$EC2_ROLE_NAME" || true
    fi

    echo "Cleanup completed."
}

# Generate unique identifiers for resources
TIMESTAMP=$(date +%Y%m%d%H%M%S)
FIS_ROLE_NAME="FISRole-${TIMESTAMP}"
FIS_POLICY_NAME="FISPolicy-${TIMESTAMP}"
```

```
EC2_ROLE_NAME="EC2SSMRole-${TIMESTAMP}"
INSTANCE_PROFILE_NAME="EC2SSMProfile-${TIMESTAMP}"
ALARM_NAME="FIS-CPU-Alarm-${TIMESTAMP}"

# Track created resources
CREATED_RESOURCES=()

echo "Step 1: Creating IAM role for AWS FIS"
# Create trust policy file for AWS FIS
cat > fis-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "fis.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role for FIS
echo "Creating IAM role $FIS_ROLE_NAME for AWS FIS..."
FIS_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$FIS_ROLE_NAME" \
  --assume-role-policy-document file://fis-trust-policy.json)
check_error "$FIS_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$FIS_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $FIS_ROLE_NAME")

# Create policy document for SSM actions
cat > fis-ssm-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:SendCommand",
        "ssm:ListCommands",
```

```
        "ssm:ListCommandInvocations"
      ],
      "Resource": "*"
    }
  ]
}
EOF

# Attach policy to the role
echo "Attaching policy $FIS_POLICY_NAME to role $FIS_ROLE_NAME..."
FIS_POLICY_OUTPUT=$(aws iam put-role-policy \
  --role-name "$FIS_ROLE_NAME" \
  --policy-name "$FIS_POLICY_NAME" \
  --policy-document file://fis-ssm-policy.json)
check_error "$FIS_POLICY_OUTPUT" "aws iam put-role-policy"
CREATED_RESOURCES+=("IAM Policy: $FIS_POLICY_NAME attached to $FIS_ROLE_NAME")

echo "Step 2: Creating IAM role for EC2 instance with SSM permissions"
# Create trust policy file for EC2
cat > ec2-trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role for EC2
echo "Creating IAM role $EC2_ROLE_NAME for EC2 instance..."
EC2_ROLE_OUTPUT=$(aws iam create-role \
  --role-name "$EC2_ROLE_NAME" \
  --assume-role-policy-document file://ec2-trust-policy.json)
check_error "$EC2_ROLE_OUTPUT" "aws iam create-role"
aws iam tag-role --role-name "$EC2_ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-fault-injection-service-gs
CREATED_RESOURCES+=("IAM Role: $EC2_ROLE_NAME")
```

```
# Attach SSM policy to the EC2 role
echo "Attaching AmazonSSMManagedInstanceCore policy to role $EC2_ROLE_NAME..."
EC2_POLICY_OUTPUT=$(aws iam attach-role-policy \
  --role-name "$EC2_ROLE_NAME" \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore)
check_error "$EC2_POLICY_OUTPUT" "aws iam attach-role-policy"
CREATED_RESOURCES+=("IAM Policy: AmazonSSMManagedInstanceCore attached to
 $EC2_ROLE_NAME")

# Create instance profile
echo "Creating instance profile $INSTANCE_PROFILE_NAME..."
PROFILE_OUTPUT=$(aws iam create-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME")
check_error "$PROFILE_OUTPUT" "aws iam create-instance-profile"
CREATED_RESOURCES+=("IAM Instance Profile: $INSTANCE_PROFILE_NAME")

# Add role to instance profile
echo "Adding role $EC2_ROLE_NAME to instance profile $INSTANCE_PROFILE_NAME..."
ADD_ROLE_OUTPUT=$(aws iam add-role-to-instance-profile \
  --instance-profile-name "$INSTANCE_PROFILE_NAME" \
  --role-name "$EC2_ROLE_NAME")
check_error "$ADD_ROLE_OUTPUT" "aws iam add-role-to-instance-profile"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

echo "Step 3: Launching EC2 instance"
# Get the latest Amazon Linux 2 AMI ID
echo "Finding latest Amazon Linux 2 AMI..."
AMI_ID=$(aws ec2 describe-images \
  --owners amazon \
  --filters "Name=name,Values=amzn2-ami-hvm-*-x86_64-gp2"
 "Name=state,Values=available" \
  --query "sort_by(Images, &CreationDate)[-1].ImageId" \
  --output text)
check_error "$AMI_ID" "aws ec2 describe-images"
echo "Using AMI: $AMI_ID"

# Launch EC2 instance
echo "Launching EC2 instance with AMI $AMI_ID..."
INSTANCE_OUTPUT=$(aws ec2 run-instances \
  --image-id "$AMI_ID" \
  --instance-type t2.micro \
```

```
  --iam-instance-profile Name="$INSTANCE_PROFILE_NAME" \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=FIS-Test-
Instance},{Key=project,Value=doc-smith},{Key=tutorial,Value=aws-fault-injection-
service-gs}]')
check_error "$INSTANCE_OUTPUT" "aws ec2 run-instances"

# Get instance ID
INSTANCE_ID=$(echo "$INSTANCE_OUTPUT" | grep -i "InstanceId" | head -1 | awk -
F'"' '{print $4}')
if [ -z "$INSTANCE_ID" ]; then
    echo "Failed to get instance ID"
    cleanup_on_error
    exit 1
fi
echo "Launched instance: $INSTANCE_ID"
CREATED_RESOURCES+=("EC2 Instance: $INSTANCE_ID")

# Enable detailed monitoring
echo "Enabling detailed monitoring for instance $INSTANCE_ID..."
MONITOR_OUTPUT=$(aws ec2 monitor-instances --instance-ids "$INSTANCE_ID")
check_error "$MONITOR_OUTPUT" "aws ec2 monitor-instances"

# Wait for instance to be running and status checks to pass
echo "Waiting for instance to be ready..."
aws ec2 wait instance-running --instance-ids "$INSTANCE_ID"
aws ec2 wait instance-status-ok --instance-ids "$INSTANCE_ID"
echo "Instance is ready"

echo "Step 4: Creating CloudWatch alarm for CPU utilization"
# Create CloudWatch alarm
echo "Creating CloudWatch alarm $ALARM_NAME..."
ALARM_OUTPUT=$(aws cloudwatch put-metric-alarm \
  --alarm-name "$ALARM_NAME" \
  --alarm-description "Alarm when CPU exceeds 50%" \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Maximum \
  --period 60 \
  --threshold 50 \
  --comparison-operator GreaterThanOrEqualToThreshold \
  --dimensions "Name=InstanceId,Value=$INSTANCE_ID" \
  --evaluation-periods 1)
check_error "$ALARM_OUTPUT" "aws cloudwatch put-metric-alarm"
CREATED_RESOURCES+=("CloudWatch Alarm: $ALARM_NAME")
```

```
# Get the alarm ARN
echo "Getting CloudWatch alarm ARN..."
ALARM_ARN_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
check_error "$ALARM_ARN_OUTPUT" "aws cloudwatch describe-alarms"
ALARM_ARN=$(echo "$ALARM_ARN_OUTPUT" | grep -i "AlarmArn" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$ALARM_ARN" ]; then
    echo "Failed to get alarm ARN"
    cleanup_on_error
    exit 1
fi
echo "Alarm ARN: $ALARM_ARN"

# Wait for the alarm to initialize and reach OK state
echo "Waiting for CloudWatch alarm to initialize (60 seconds)..."
sleep 60

# Check alarm state
echo "Checking alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
  --alarm-names "$ALARM_NAME")
ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 | awk -
F'"' '{print $4}')
echo "Current alarm state: $ALARM_STATE"

# If alarm is not in OK state, wait longer or generate some baseline metrics
if [ "$ALARM_STATE" != "OK" ]; then
    echo "Alarm not in OK state. Waiting for alarm to stabilize (additional 60
 seconds)..."
    sleep 60

    # Check alarm state again
    ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms \
      --alarm-names "$ALARM_NAME")
    ALARM_STATE=$(echo "$ALARM_STATE_OUTPUT" | grep -i "StateValue" | head -1 |
 awk -F'"' '{print $4}')
    echo "Updated alarm state: $ALARM_STATE"

    if [ "$ALARM_STATE" != "OK" ]; then
        echo "Warning: Alarm still not in OK state. Experiment may fail to
 start."
    fi
```

```
fi

echo "Step 5: Creating AWS FIS experiment template"
# Get the IAM role ARN
echo "Getting IAM role ARN for $FIS_ROLE_NAME..."
ROLE_ARN_OUTPUT=$(aws iam get-role \
  --role-name "$FIS_ROLE_NAME")
check_error "$ROLE_ARN_OUTPUT" "aws iam get-role"
ROLE_ARN=$(echo "$ROLE_ARN_OUTPUT" | grep -i "Arn" | head -1 | awk -F'"' '{print
 $4}')
if [ -z "$ROLE_ARN" ]; then
    echo "Failed to get role ARN"
    cleanup_on_error
    exit 1
fi
echo "Role ARN: $ROLE_ARN"

# Get account ID and region
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
REGION=$(aws configure get region)
if [ -z "$REGION" ]; then
    REGION="us-east-1"  # Default to us-east-1 if region not set
fi
INSTANCE_ARN="arn:aws:ec2:${REGION}:${ACCOUNT_ID}:instance/${INSTANCE_ID}"
echo "Instance ARN: $INSTANCE_ARN"

# Create experiment template - Fixed JSON escaping issue
cat > experiment-template.json << EOF
{
  "description": "Test CPU stress predefined SSM document",
  "targets": {
    "testInstance": {
      "resourceType": "aws:ec2:instance",
      "resourceArns": ["$INSTANCE_ARN"],
      "selectionMode": "ALL"
    }
  },
  "actions": {
    "runCpuStress": {
      "actionId": "aws:ssm:send-command",
      "parameters": {
        "documentArn": "arn:aws:ssm:$REGION::document/AWSFIS-Run-CPU-Stress",
        "documentParameters": "{\"DurationSeconds\":\"120\"}",
        "duration": "PT5M"
```

```
      },
      "targets": {
        "Instances": "testInstance"
      }
    }
  },
  "stopConditions": [
    {
      "source": "aws:cloudwatch:alarm",
      "value": "$ALARM_ARN"
    }
  ],
  "roleArn": "$ROLE_ARN",
  "tags": {
    "Name": "FIS-CPU-Stress-Experiment",
    "project": "doc-smith",
    "tutorial": "aws-fault-injection-service-gs"
  }
}
EOF

# Create experiment template
echo "Creating AWS FIS experiment template..."
TEMPLATE_OUTPUT=$(aws fis create-experiment-template --cli-input-json file://
experiment-template.json)
check_error "$TEMPLATE_OUTPUT" "aws fis create-experiment-template"
TEMPLATE_ID=$(echo "$TEMPLATE_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$TEMPLATE_ID" ]; then
    echo "Failed to get template ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment template created with ID: $TEMPLATE_ID"
CREATED_RESOURCES+=("FIS Experiment Template: $TEMPLATE_ID")

echo "Step 6: Starting the experiment"
# Start the experiment
echo "Starting AWS FIS experiment using template $TEMPLATE_ID..."
EXPERIMENT_OUTPUT=$(aws fis start-experiment \
  --experiment-template-id "$TEMPLATE_ID" \
  --tags '{"Name": "FIS-CPU-Stress-Run"}')
check_error "$EXPERIMENT_OUTPUT" "aws fis start-experiment"
```

```
EXPERIMENT_ID=$(echo "$EXPERIMENT_OUTPUT" | grep -i "id" | head -1 | awk -F'"'
 '{print $4}')
if [ -z "$EXPERIMENT_ID" ]; then
    echo "Failed to get experiment ID"
    cleanup_on_error
    exit 1
fi
echo "Experiment started with ID: $EXPERIMENT_ID"
CREATED_RESOURCES+=("FIS Experiment: $EXPERIMENT_ID")

echo "Step 7: Tracking experiment progress"
# Track experiment progress
echo "Tracking experiment progress..."
MAX_CHECKS=30
CHECK_COUNT=0
EXPERIMENT_STATE=""

while [ $CHECK_COUNT -lt $MAX_CHECKS ]; do
    EXPERIMENT_INFO=$(aws fis get-experiment --id "$EXPERIMENT_ID")
    # Don't check for errors here, as we expect some experiments to fail

    EXPERIMENT_STATE=$(echo "$EXPERIMENT_INFO" | grep -i "status" | head -1 | awk
 -F'"' '{print $4}')
    echo "Experiment state: $EXPERIMENT_STATE"

    if [ "$EXPERIMENT_STATE" == "completed" ] || [ "$EXPERIMENT_STATE" ==
 "stopped" ] || [ "$EXPERIMENT_STATE" == "failed" ]; then
        # Show the reason for the state
        REASON=$(echo "$EXPERIMENT_INFO" | grep -i "reason" | head -1 | awk -F'"'
 '{print $4}')
        if [ -n "$REASON" ]; then
            echo "Reason: $REASON"
        fi
        break
    fi

    echo "Waiting 10 seconds before checking again..."
    sleep 10
    CHECK_COUNT=$((CHECK_COUNT + 1))
done

if [ $CHECK_COUNT -eq $MAX_CHECKS ]; then
    echo "Experiment is taking longer than expected. You can check its status
 later using:"
```

```
    echo "aws fis get-experiment --id $EXPERIMENT_ID"
fi

echo "Step 8: Verifying experiment results"
# Check CloudWatch alarm state
echo "Checking CloudWatch alarm state..."
ALARM_STATE_OUTPUT=$(aws cloudwatch describe-alarms --alarm-names "$ALARM_NAME")
check_error "$ALARM_STATE_OUTPUT" "aws cloudwatch describe-alarms"
echo "$ALARM_STATE_OUTPUT"

# Get CPU utilization metrics
echo "Getting CPU utilization metrics..."
END_TIME=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

# FIXED: Cross-platform compatible way to calculate time 10 minutes ago
# This approach uses epoch seconds and basic arithmetic which works on all Linux
 distributions
CURRENT_EPOCH=$(date +%s)
TEN_MINUTES_AGO_EPOCH=$((CURRENT_EPOCH - 600))
START_TIME=$(date -u -d "@$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ" 2>/dev/
null || date -u -r "$TEN_MINUTES_AGO_EPOCH" +"%Y-%m-%dT%H:%M:%SZ")

# Create metric query file
cat > metric-query.json << EOF
[
  {
    "Id": "cpu",
    "MetricStat": {
      "Metric": {
        "Namespace": "AWS/EC2",
        "MetricName": "CPUUtilization",
        "Dimensions": [
          {
            "Name": "InstanceId",
            "Value": "$INSTANCE_ID"
          }
        ]
      },
      "Period": 60,
      "Stat": "Maximum"
    }
  }
]
EOF
```

```
METRICS_OUTPUT=$(aws cloudwatch get-metric-data \
  --start-time "$START_TIME" \
  --end-time "$END_TIME" \
  --metric-data-queries file://metric-query.json)
check_error "$METRICS_OUTPUT" "aws cloudwatch get-metric-data"
echo "CPU Utilization Metrics:"
echo "$METRICS_OUTPUT"

# Display summary of created resources
echo ""
echo "==========================================="
echo "RESOURCES CREATED"
echo "==========================================="
for resource in "${CREATED_RESOURCES[@]}"; do
    echo "- $resource"
done
echo "==========================================="

# Prompt for cleanup
echo ""
echo "==========================================="
echo "CLEANUP CONFIRMATION"
echo "==========================================="
echo "Do you want to clean up all created resources? (y/n): "
read -r CLEANUP_CHOICE

if [[ "$CLEANUP_CHOICE" =~ ^[Yy]$ ]]; then
    echo "Starting cleanup process..."

    # Stop experiment if still running
    if [ "$EXPERIMENT_STATE" != "completed" ] && [ "$EXPERIMENT_STATE" !=
 "stopped" ] && [ "$EXPERIMENT_STATE" != "failed" ]; then
        echo "Stopping experiment $EXPERIMENT_ID..."
        STOP_OUTPUT=$(aws fis stop-experiment --id "$EXPERIMENT_ID")
        check_error "$STOP_OUTPUT" "aws fis stop-experiment"
        echo "Waiting for experiment to stop..."
        sleep 10
    fi

    # Delete experiment template
    echo "Deleting experiment template $TEMPLATE_ID..."
    DELETE_TEMPLATE_OUTPUT=$(aws fis delete-experiment-template --id
 "$TEMPLATE_ID")
```

```
    check_error "$DELETE_TEMPLATE_OUTPUT" "aws fis delete-experiment-template"

    # Delete CloudWatch alarm
    echo "Deleting CloudWatch alarm $ALARM_NAME..."
    DELETE_ALARM_OUTPUT=$(aws cloudwatch delete-alarms --alarm-names
"$ALARM_NAME")
    check_error "$DELETE_ALARM_OUTPUT" "aws cloudwatch delete-alarms"

    # Terminate EC2 instance
    echo "Terminating EC2 instance $INSTANCE_ID..."
    TERMINATE_OUTPUT=$(aws ec2 terminate-instances --instance-ids "$INSTANCE_ID")
    check_error "$TERMINATE_OUTPUT" "aws ec2 terminate-instances"
    echo "Waiting for instance to terminate..."
    aws ec2 wait instance-terminated --instance-ids "$INSTANCE_ID"

    # Clean up IAM resources
    echo "Removing role from instance profile..."
    REMOVE_ROLE_OUTPUT=$(aws iam remove-role-from-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME" \
      --role-name "$EC2_ROLE_NAME")
    check_error "$REMOVE_ROLE_OUTPUT" "aws iam remove-role-from-instance-profile"

    echo "Deleting instance profile..."
    DELETE_PROFILE_OUTPUT=$(aws iam delete-instance-profile \
      --instance-profile-name "$INSTANCE_PROFILE_NAME")
    check_error "$DELETE_PROFILE_OUTPUT" "aws iam delete-instance-profile"

    echo "Deleting FIS role policy..."
    DELETE_POLICY_OUTPUT=$(aws iam delete-role-policy \
      --role-name "$FIS_ROLE_NAME" \
      --policy-name "$FIS_POLICY_NAME")
    check_error "$DELETE_POLICY_OUTPUT" "aws iam delete-role-policy"

    echo "Detaching policy from EC2 role..."
    DETACH_POLICY_OUTPUT=$(aws iam detach-role-policy \
      --role-name "$EC2_ROLE_NAME" \
      --policy-arn "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore")
    check_error "$DETACH_POLICY_OUTPUT" "aws iam detach-role-policy"

    echo "Deleting FIS role..."
    DELETE_FIS_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$FIS_ROLE_NAME")
    check_error "$DELETE_FIS_ROLE_OUTPUT" "aws iam delete-role"
```

```
    echo "Deleting EC2 role..."
    DELETE_EC2_ROLE_OUTPUT=$(aws iam delete-role \
      --role-name "$EC2_ROLE_NAME")
    check_error "$DELETE_EC2_ROLE_OUTPUT" "aws iam delete-role"

    # Clean up temporary files
    echo "Cleaning up temporary files..."
    rm -f fis-trust-policy.json ec2-trust-policy.json fis-ssm-policy.json
 experiment-template.json metric-query.json

    echo "Cleanup completed successfully."
else
    echo "Cleanup skipped. Resources will remain in your AWS account."
    echo "You can manually clean up the resources listed above."
fi

echo ""
echo "Script execution completed."
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AddRoleToInstanceProfile
  - AttachRolePolicy
  - CreateExperimentTemplate
  - CreateInstanceProfile
  - CreateRole
  - DeleteAlarms
  - DeleteExperimentTemplate
  - DeleteInstanceProfile
  - DeleteRole
  - DeleteRolePolicy
  - DescribeAlarms
  - DescribeImages
  - DetachRolePolicy
  - GetCallerIdentity
  - GetExperiment

- GetMetricData
- GetRole
- MonitorInstances
- PutMetricAlarm
- PutRolePolicy
- RemoveRoleFromInstanceProfile
- RunInstances
- StartExperiment
- StopExperiment
- TerminateInstances
- Wait

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Send OpenTelemetry metrics to CloudWatch and alarm on them with PromQL using an AWS SDK

The following code examples show how to:

- Send OTLP metrics to the CloudWatch metrics endpoint with an OpenTelemetry Collector.
- Start OpenTelemetry enrichment so CloudWatch correlates those metrics with your resources.
- Create an alarm that evaluates a PromQL query across every series the query returns.
- Inspect the individual series, called contributors, that put the alarm in ALARM state.
- Mute the alarm for a maintenance window, then clean up.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Wrapper code for the CloudWatch OpenTelemetry operations.

```
/// <summary>
/// Wrapper class for the OpenTelemetry features of Amazon CloudWatch: turning on
 OTel
/// enrichment so that CloudWatch vended metrics are queryable with PromQL,
 alarming on a
/// PromQL query, inspecting the individual series (contributors) that put a
 PromQL alarm
/// into ALARM, and muting alarm actions on a schedule.
///
/// Note that OTLP metric ingestion is not an AWS SDK operation. To send
 OpenTelemetry
/// metrics to CloudWatch, point an OpenTelemetry collector or the AWS Distro for
/// OpenTelemetry (ADOT) SDK at the CloudWatch OTLP metrics endpoint,
/// https://monitoring.{region}.amazonaws.com/v1/metrics. The operations here
 cover
/// everything you do after those metrics land in CloudWatch.
/// </summary>
public class CloudWatchOTelWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchOTelWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch OpenTelemetry wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchOTelWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchOTelWrapper> logger)
    {
        _logger = logger;
```

```
        _amazonCloudWatch = amazonCloudWatch;
    }


    /// <summary>
    /// Turn on OTel enrichment for the account. Once enrichment is running,
CloudWatch
    /// vended metrics that carry a resource identifier dimension, such as the
Amazon EC2
    /// CPUUtilization metric with its InstanceId dimension, are decorated with
resource
    /// ARN and resource tag labels and become queryable with PromQL.
    ///
    /// Resource tags on telemetry must already be enabled for the account before
you
    /// call this operation.
    /// </summary>
    /// <returns>True if successful.</returns>
    public async Task<bool> StartOTelEnrichment()
    {
        var response = await _amazonCloudWatch.StartOTelEnrichmentAsync(
            new StartOTelEnrichmentRequest());

        _logger.LogInformation("Started OTel enrichment for this account.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Get the current OTel enrichment status for the account.
    /// </summary>
    /// <returns>The status, either Running or Stopped.</returns>
    public async Task<OTelEnrichmentStatus> GetOTelEnrichmentStatus()
    {
        var response = await _amazonCloudWatch.GetOTelEnrichmentAsync(
            new GetOTelEnrichmentRequest());

        _logger.LogInformation($"OTel enrichment status is {response.Status}.");
        return response.Status;
    }

    /// <summary>
    /// Create an alarm that evaluates a PromQL query.
    ///
```

```
   /// A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
   /// match many series at once, and each matching series is tracked separately
as a
   /// contributor. Instead of counting breaching periods, you specify
durations: a
   /// contributor moves to ALARM after it breaches continuously for the pending
period,
   /// and back to OK after it stops breaching for the recovery period. A PromQL
alarm
   /// starts in the OK state rather than INSUFFICIENT_DATA.
   ///
   /// EvaluationCriteria is a union and is mutually exclusive with the classic
   /// MetricName and Metrics properties. When you use it you must also set
   /// EvaluationInterval, and you must not set Period, Statistic, Threshold,
   /// ComparisonOperator, EvaluationPeriods, DatapointsToAlarm, or
TreatMissingData.
   /// </summary>
   /// <param name="alarmName">The name of the alarm, unique within the
Region.</param>
   /// <param name="query">The PromQL query to evaluate, such as
   /// avg(cpu_utilization_percent) &gt; 80. The comparison belongs in the query
itself;
   /// there is no separate threshold property.</param>
   /// <param name="evaluationInterval">How often, in seconds, to run the query.
Valid
   /// values are 10, 20, 30, and any multiple of 60, up to 3600.</param>
   /// <param name="pendingPeriod">How long, in seconds, a contributor must
breach
   /// continuously before it moves to ALARM.</param>
   /// <param name="recoveryPeriod">How long, in seconds, a contributor must
stop
   /// breaching before it moves back to OK.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutPromQLMetricAlarm(string alarmName, string query,
       int evaluationInterval = 60, int pendingPeriod = 300, int recoveryPeriod
= 120)
   {
       var response = await _amazonCloudWatch.PutMetricAlarmAsync(
           new PutMetricAlarmRequest
           {
               AlarmName = alarmName,
               AlarmDescription = "A PromQL alarm created by the AWS SDK
for .NET example.",
```

```
                EvaluationCriteria = new EvaluationCriteria
                {
                    PromQLCriteria = new AlarmPromQLCriteria
                    {
                        Query = query,
                        PendingPeriod = pendingPeriod,
                        RecoveryPeriod = recoveryPeriod
                    }
                },
                EvaluationInterval = evaluationInterval
            });

        _logger.LogInformation($"Created PromQL alarm {alarmName} for query
{query}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Get the contributors for a PromQL alarm. Each contributor is one series
that the
    /// alarm's query matched, identified by its label set. This is how you find
out which
    /// hosts, services, or pods are breaching, rather than only that something
is.
    /// </summary>
    /// <param name="alarmName">The name of the PromQL alarm.</param>
    /// <returns>The list of contributors.</returns>
    public async Task<List<AlarmContributor>> DescribeAlarmContributors(string
alarmName)
    {
        var results = new List<AlarmContributor>();
        string? nextToken = null;

        do
        {
            var response = await
_amazonCloudWatch.DescribeAlarmContributorsAsync(
                new DescribeAlarmContributorsRequest
                {
                    AlarmName = alarmName,
                    NextToken = nextToken
                });

            if (response.AlarmContributors != null)
```

```
            {
                results.AddRange(response.AlarmContributors);
            }

            nextToken = response.NextToken;
        } while (!string.IsNullOrEmpty(nextToken));

        _logger.LogInformation($"Got {results.Count} contributors for alarm
 {alarmName}.");
        return results;
    }

    /// <summary>
    /// Create or update an alarm mute rule. While a mute rule is active the
 targeted
    /// alarms keep evaluating and keep transitioning between states, but their
 configured
    /// actions do not fire. This is the supported way to suppress notifications
 during a
    /// known maintenance window, instead of disabling alarm actions and relying
 on
    /// someone to turn them back on.
    /// </summary>
    /// <param name="name">The name of the mute rule.</param>
    /// <param name="expression">When the rule activates. For a recurring window,
 use a
    /// five-field cron expression, cron(Minutes Hours Day-of-month Month Day-of-
week),
    /// such as cron(0 2 * * SUN) for every Sunday at 2:00 AM. Note that this is
 five
    /// fields, not the six that Amazon EventBridge uses. For a one-time window,
 use
    /// at(yyyy-MM-ddThh:mm), such as at(2026-09-05T02:00).</param>
    /// <param name="duration">How long the mute window lasts once it activates,
 in
    /// ISO 8601 duration format, from PT1M (one minute) to P15D (15 days). For
 example,
    /// PT2H is two hours and P2DT12H is two days and 12 hours.</param>
    /// <param name="timezone">The time zone the expression is evaluated in, such
 as
    /// America/Los_Angeles.</param>
    /// <param name="alarmNames">The names of up to 100 alarms to mute. If null
 or empty,
    /// the rule applies to all alarms in the account.</param>
```

```
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAlarmMuteRule(string name, string expression,
string duration,
        string timezone, List<string>? alarmNames = null)
    {
        var request = new PutAlarmMuteRuleRequest
        {
            Name = name,
            Description = "A mute rule created by the AWS SDK for .NET example.",
            Rule = new Rule
            {
                Schedule = new Schedule
                {
                    Expression = expression,
                    Duration = duration,
                    Timezone = timezone
                }
            }
        };

        if (alarmNames != null && alarmNames.Any())
        {
            request.MuteTargets = new MuteTargets { AlarmNames = alarmNames };
        }

        var response = await _amazonCloudWatch.PutAlarmMuteRuleAsync(request);

        _logger.LogInformation($"Put alarm mute rule {name}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Get the full configuration of an alarm mute rule, including its schedule,
the
    /// alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.
    /// </summary>
    /// <param name="name">The name of the mute rule.</param>
    /// <returns>The mute rule.</returns>
    public async Task<GetAlarmMuteRuleResponse> GetAlarmMuteRule(string name)
    {
        var response = await _amazonCloudWatch.GetAlarmMuteRuleAsync(
            new GetAlarmMuteRuleRequest
            {
```

```
                AlarmMuteRuleName = name
            });

        _logger.LogInformation($"Mute rule {response.Name} is
{response.Status}.");
        return response;
    }

    /// <summary>
    /// List the alarm mute rules in the account, optionally filtered to the
rules that
    /// target one alarm.
    /// </summary>
    /// <param name="alarmName">When specified, only rules that target this alarm
are
    /// returned.</param>
    /// <returns>The list of mute rule summaries.</returns>
    public async Task<List<AlarmMuteRuleSummary>> ListAlarmMuteRules(string?
alarmName = null)
    {
        var results = new List<AlarmMuteRuleSummary>();
        string? nextToken = null;

        do
        {
            var response = await _amazonCloudWatch.ListAlarmMuteRulesAsync(
                new ListAlarmMuteRulesRequest
                {
                    AlarmName = alarmName,
                    NextToken = nextToken
                });

            if (response.AlarmMuteRuleSummaries != null)
            {
                results.AddRange(response.AlarmMuteRuleSummaries);
            }

            nextToken = response.NextToken;
        } while (!string.IsNullOrEmpty(nextToken));

        _logger.LogInformation($"Got {results.Count} alarm mute rules.");
        return results;
    }
```

```
    /// <summary>
    /// Delete an alarm mute rule. The alarms it targeted resume firing their
actions.
    /// </summary>
    /// <param name="name">The name of the mute rule.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarmMuteRule(string name)
    {
        var response = await _amazonCloudWatch.DeleteAlarmMuteRuleAsync(
            new DeleteAlarmMuteRuleRequest
            {
                AlarmMuteRuleName = name
            });

        _logger.LogInformation($"Deleted alarm mute rule {name}.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Turn off OTel enrichment for the account. Existing PromQL alarms are not
deleted,
    /// but vended metrics stop being enriched with resource ARN and tag labels,
so
    /// queries that select on those labels stop matching.
    /// </summary>
    /// <returns>True if successful.</returns>
    public async Task<bool> StopOTelEnrichment()
    {
        var response = await _amazonCloudWatch.StopOTelEnrichmentAsync(
            new StopOTelEnrichmentRequest());

        _logger.LogInformation("Stopped OTel enrichment for this account.");
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DescribeAlarmContributors
  - GetAlarmMuteRule

- GetOTelEnrichment
- ListAlarmMuteRules
- PutAlarmMuteRule
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

A wrapper class for the CloudWatch OpenTelemetry SDK operations.

```
import software.amazon.awssdk.services.cloudwatch.CloudWatchClient;
import software.amazon.awssdk.services.cloudwatch.model.AlarmContributor;
import software.amazon.awssdk.services.cloudwatch.model.AlarmMuteRuleSummary;
import software.amazon.awssdk.services.cloudwatch.model.AlarmPromQLCriteria;
import software.amazon.awssdk.services.cloudwatch.model.CloudWatchException;
import
 software.amazon.awssdk.services.cloudwatch.model.DeleteAlarmMuteRuleRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmContributorsRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.DescribeAlarmContributorsResponse;
import software.amazon.awssdk.services.cloudwatch.model.EvaluationCriteria;
import software.amazon.awssdk.services.cloudwatch.model.GetAlarmMuteRuleRequest;
import software.amazon.awssdk.services.cloudwatch.model.GetAlarmMuteRuleResponse;
import software.amazon.awssdk.services.cloudwatch.model.GetOTelEnrichmentRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.ListAlarmMuteRulesRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.ListAlarmMuteRulesResponse;
import software.amazon.awssdk.services.cloudwatch.model.MuteTargets;
```

```
import software.amazon.awssdk.services.cloudwatch.model.PutAlarmMuteRuleRequest;
import software.amazon.awssdk.services.cloudwatch.model.PutMetricAlarmRequest;
import software.amazon.awssdk.services.cloudwatch.model.Rule;
import software.amazon.awssdk.services.cloudwatch.model.Schedule;
import
 software.amazon.awssdk.services.cloudwatch.model.StartOTelEnrichmentRequest;
import
 software.amazon.awssdk.services.cloudwatch.model.StopOTelEnrichmentRequest;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class CloudWatchOTelActions {


    /**
     * Turns on OTel enrichment for the account. Once enrichment is running,
 CloudWatch
     * vended metrics that carry a resource identifier dimension, such as the EC2
     * CPUUtilization metric with its InstanceId dimension, are decorated with
 resource
     * ARN and resource tag labels and become queryable with PromQL.
     *
     * <p>Resource tags on telemetry must already be enabled for the account
 before you
     * call this operation.
     *
     * @param cw the CloudWatch client
     */
    public static void startOTelEnrichment(CloudWatchClient cw) {
        try {
            cw.startOTelEnrichment(StartOTelEnrichmentRequest.builder().build());
            System.out.println("Started OTel enrichment for this account.");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }


    /**
     * Gets the current OTel enrichment status for the account.
```

```
     *
     * @param cw the CloudWatch client
     * @return the status, either {@code Running} or {@code Stopped}
     */
    public static String getOTelEnrichmentStatus(CloudWatchClient cw) {
        try {
            String status =
cw.getOTelEnrichment(GetOTelEnrichmentRequest.builder().build())
                    .statusAsString();
            System.out.printf("OTel enrichment status is %s.%n", status);
            return status;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return null;
        }
    }


    /**
     * Turns off OTel enrichment for the account. Existing PromQL alarms are not
     * deleted, but vended metrics stop being enriched with resource ARN and tag
labels,
     * so queries that select on those labels stop matching.
     *
     * @param cw the CloudWatch client
     */
    public static void stopOTelEnrichment(CloudWatchClient cw) {
        try {
            cw.stopOTelEnrichment(StopOTelEnrichmentRequest.builder().build());
            System.out.println("Stopped OTel enrichment for this account.");

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }


    /**
     * Creates an alarm that evaluates a PromQL query.
     *
```

```
    * <p>A PromQL alarm differs from a classic metric alarm in a few ways. The
query
    * can match many series at once, and each matching series is tracked
separately as
    * a contributor. Instead of counting breaching periods, you specify
durations: a
    * contributor moves to ALARM after it breaches continuously for the pending
period,
    * and back to OK after it stops breaching for the recovery period. A PromQL
alarm
    * starts in the OK state rather than INSUFFICIENT_DATA.
    *
    * <p>{@link EvaluationCriteria} is a union and is mutually exclusive with
the
    * classic {@code metricName} and {@code metrics} parameters. When you use it
you
    * must also set {@code evaluationInterval}, and you must not set {@code
period},
    * {@code statistic}, {@code threshold}, {@code comparisonOperator},
    * {@code evaluationPeriods}, {@code datapointsToAlarm}, or
    * {@code treatMissingData}.
    *
    * @param cw                 the CloudWatch client
    * @param alarmName          the name of the alarm, unique within the Region
    * @param query              the PromQL query to evaluate, such as
    *                           {@code avg(cpu_utilization_percent) > 80}. The
    *                           comparison belongs in the query itself; there is
no
    *                           separate threshold parameter.
    * @param evaluationInterval how often, in seconds, to run the query. Valid
values
    *                           are 10, 20, 30, and any multiple of 60, up to
3600.
    * @param pendingPeriod      how long, in seconds, a contributor must breach
    *                           continuously before it moves to ALARM
    * @param recoveryPeriod     how long, in seconds, a contributor must stop
breaching
    *                           before it moves back to OK
    */
   public static void putPromQLMetricAlarm(CloudWatchClient cw, String
alarmName, String query,
           int evaluationInterval, int pendingPeriod, int recoveryPeriod) {
       try {
           AlarmPromQLCriteria promQLCriteria = AlarmPromQLCriteria.builder()
```

```
                    .query(query)
                    .pendingPeriod(pendingPeriod)
                    .recoveryPeriod(recoveryPeriod)
                    .build();

            PutMetricAlarmRequest request = PutMetricAlarmRequest.builder()
                    .alarmName(alarmName)
                    .alarmDescription("PromQL alarm created by the AWS SDK for
Java 2.x example.")
                    .evaluationCriteria(EvaluationCriteria.builder()
                            .promQLCriteria(promQLCriteria)
                            .build())
                    .evaluationInterval(evaluationInterval)
                    .build();

            cw.putMetricAlarm(request);
            System.out.printf("Created PromQL alarm %s for query %s.%n",
alarmName, query);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }


    /**
     * Gets the contributors for a PromQL alarm. Each contributor is one series
that the
     * alarm's query matched, identified by its label set. This is how you find
out which
     * hosts, services, or pods are breaching, rather than only that something
is.
     *
     * @param cw        the CloudWatch client
     * @param alarmName the name of the PromQL alarm
     * @return the list of contributors
     */
    public static List<AlarmContributor>
describeAlarmContributors(CloudWatchClient cw, String alarmName) {
        List<AlarmContributor> contributors = new ArrayList<>();
        try {
            String nextToken = null;
            do {
```

```
                DescribeAlarmContributorsRequest request =
DescribeAlarmContributorsRequest.builder()
                        .alarmName(alarmName)
                        .nextToken(nextToken)
                        .build();

                DescribeAlarmContributorsResponse response =
cw.describeAlarmContributors(request);
                contributors.addAll(response.alarmContributors());
                nextToken = response.nextToken();
            } while (nextToken != null && !nextToken.isEmpty());

            for (AlarmContributor contributor : contributors) {
                StringBuilder labels = new StringBuilder();
                for (Map.Entry<String, String> attribute :
contributor.contributorAttributes().entrySet()) {
                    if (labels.length() > 0) {
                        labels.append(", ");
                    }

labels.append(attribute.getKey()).append("=").append(attribute.getValue());
                }
                System.out.printf("%s: %s%n", contributor.contributorId(),
labels);
                System.out.printf("  reason: %s%n", contributor.stateReason());
            }
            return contributors;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return contributors;
        }
    }


    /**
     * Creates or updates an alarm mute rule. While a mute rule is active the
targeted
     * alarms keep evaluating and keep transitioning between states, but their
configured
     * actions do not fire. This is the supported way to suppress notifications
during a
```

```
     * known maintenance window instead of disabling alarm actions and hoping
 someone
     * remembers to turn them back on.
     *
     * @param cw         the CloudWatch client
     * @param name       the name of the mute rule
     * @param expression when the rule activates. For a recurring window, use a
     *                   five-field cron expression,
     *                   {@code cron(Minutes Hours Day-of-month Month Day-of-
week)},
     *                   such as {@code cron(0 2 * * SUN)} for every Sunday at
 2:00 AM.
     *                   Note that this is five fields, not the six that Amazon
     *                   EventBridge uses. For a one-time window, use
     *                   {@code at(yyyy-MM-ddThh:mm)}, such as
     *                   {@code at(2026-09-05T02:00)}.
     * @param duration   how long the mute window lasts once it activates, in ISO
 8601
     *                   duration format, from {@code PT1M} (one minute) to
     *                   {@code P15D} (15 days). For example, {@code PT2H} is two
 hours
     *                   and {@code P2DT12H} is two days and 12 hours.
     * @param timezone   the time zone the expression is evaluated in, such as
     *                   {@code America/Los_Angeles}
     * @param alarmNames the names of up to 100 alarms to mute. If empty, the
 rule
     *                   applies to all alarms in the account.
     */
    public static void putAlarmMuteRule(CloudWatchClient cw, String name, String
 expression, String duration,
            String timezone, List<String> alarmNames) {
        try {
            Schedule schedule = Schedule.builder()
                    .expression(expression)
                    .duration(duration)
                    .timezone(timezone)
                    .build();

            PutAlarmMuteRuleRequest.Builder request =
 PutAlarmMuteRuleRequest.builder()
                    .name(name)
                    .description("Mute rule created by the AWS SDK for Java 2.x
 example.")
                    .rule(Rule.builder().schedule(schedule).build());
```

```
            if (alarmNames != null && !alarmNames.isEmpty()) {

request.muteTargets(MuteTargets.builder().alarmNames(alarmNames).build());
            }

            cw.putAlarmMuteRule(request.build());
            System.out.printf("Put alarm mute rule %s.%n", name);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }


    /**
     * Gets the full configuration of an alarm mute rule, including its schedule,
the
     * alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.
     *
     * @param cw   the CloudWatch client
     * @param name the name of the mute rule
     * @return the mute rule
     */
    public static GetAlarmMuteRuleResponse getAlarmMuteRule(CloudWatchClient cw,
String name) {
        try {
            GetAlarmMuteRuleResponse response =
cw.getAlarmMuteRule(GetAlarmMuteRuleRequest.builder()
                    .alarmMuteRuleName(name)
                    .build());

            System.out.printf("Mute rule %s is %s.%n", response.name(),
response.statusAsString());
            return response;

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
            return null;
        }
    }
```

```
   /**
    * Lists the alarm mute rules in the account, optionally filtered to the
rules that
    * target one alarm.
    *
    * @param cw        the CloudWatch client
    * @param alarmName when non-null, only rules that target this alarm are
returned
    * @return the list of mute rule summaries
    */
   public static List<AlarmMuteRuleSummary> listAlarmMuteRules(CloudWatchClient
cw, String alarmName) {
       List<AlarmMuteRuleSummary> summaries = new ArrayList<>();
       try {
           String nextToken = null;
           do {
               ListAlarmMuteRulesRequest request =
ListAlarmMuteRulesRequest.builder()
                       .alarmName(alarmName)
                       .nextToken(nextToken)
                       .build();

               ListAlarmMuteRulesResponse response =
cw.listAlarmMuteRules(request);
               summaries.addAll(response.alarmMuteRuleSummaries());
               nextToken = response.nextToken();
           } while (nextToken != null && !nextToken.isEmpty());

           for (AlarmMuteRuleSummary summary : summaries) {
               System.out.printf("%s (%s)%n", summary.alarmMuteRuleArn(),
summary.statusAsString());
           }
           return summaries;

       } catch (CloudWatchException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
           return summaries;
       }
   }
```

```
    /**
     * Deletes an alarm mute rule. The alarms it targeted resume firing their
 actions.
     *
     * @param cw   the CloudWatch client
     * @param name the name of the mute rule
     */
    public static void deleteAlarmMuteRule(CloudWatchClient cw, String name) {
        try {
            cw.deleteAlarmMuteRule(DeleteAlarmMuteRuleRequest.builder()
                    .alarmMuteRuleName(name)
                    .build());

            System.out.printf("Deleted alarm mute rule %s.%n", name);

        } catch (CloudWatchException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetOTelEnrichment
  - ListAlarmMuteRules
  - PutAlarmMuteRule
  - PutMetricAlarm
  - StartOTelEnrichment
  - StopOTelEnrichment

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a class that wraps the CloudWatch OpenTelemetry operations.

```
import logging
import time

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class CloudWatchOTelWrapper:
    """Encapsulates the OpenTelemetry-oriented Amazon CloudWatch operations."""

    def __init__(self, cloudwatch_client):
        """
        :param cloudwatch_client: A Boto3 CloudWatch client. The OpenTelemetry
                                  operations are only available on the client
                                  interface, not on the higher-level
                                  ``boto3.resource("cloudwatch")`` interface.
        """
        self.cloudwatch_client = cloudwatch_client

    @classmethod
    def from_client(cls):
        """
        Creates a wrapper backed by a default CloudWatch client.

        :return: A CloudWatchOTelWrapper.
        """
        return cls(boto3.client("cloudwatch"))
```

```
   def start_otel_enrichment(self):
       """
       Turns on OTel enrichment for the account. Once enrichment is running,
       CloudWatch vended metrics that carry a resource identifier dimension,
such as
       the EC2 CPUUtilization metric with its InstanceId dimension, are
decorated with
       resource ARN and resource tag labels and become queryable with PromQL.

       Resource tags on telemetry must already be enabled for the account before
you
       call this operation.
       """
       try:
           # Boto3 splits the OTel prefix when it converts the
StartOTelEnrichment
           # operation name to snake case, so the method is
start_o_tel_enrichment.
           self.cloudwatch_client.start_o_tel_enrichment()
           logger.info("Started OTel enrichment for this account.")
       except ClientError:
           logger.exception("Couldn't start OTel enrichment.")
           raise


   def get_otel_enrichment_status(self):
       """
       Gets the current OTel enrichment status for the account.

       :return: The status, either 'Running' or 'Stopped'.
       """
       try:
           response = self.cloudwatch_client.get_o_tel_enrichment()
       except ClientError:
           logger.exception("Couldn't get the OTel enrichment status.")
           raise
       else:
           status = response["Status"]
           logger.info("OTel enrichment status is %s.", status)
           return status


   def create_promql_alarm(
       self,
```

```
        alarm_name,
        query,
        evaluation_interval,
        pending_period=300,
        recovery_period=120,
        description=None,
        alarm_actions=None,
    ):
        """
        Creates an alarm that evaluates a PromQL query.

        A PromQL alarm differs from a classic metric alarm in a few ways. The
query can
        match many series at once, and each matching series is tracked separately
as a
        *contributor*. Instead of counting breaching periods, you specify
durations: a
        contributor moves to ALARM after it breaches continuously for the pending
        period, and back to OK after it stops breaching for the recovery period.
A
        PromQL alarm starts in the OK state rather than INSUFFICIENT_DATA.

        The PromQL evaluation parameters live in the EvaluationCriteria union,
which is
        mutually exclusive with the classic MetricName and Metrics parameters.
When you
        use EvaluationCriteria you must also set EvaluationInterval, and you must
not
        set Period, Statistic, Threshold, ComparisonOperator, EvaluationPeriods,
        DatapointsToAlarm, or TreatMissingData.

        :param alarm_name: The name of the alarm. Must be unique within the
Region.
        :param query: The PromQL query to evaluate, such as
                      'avg(cpu_utilization_percent) > 80'. The comparison belongs
in
                      the query itself; there is no separate threshold parameter.
        :param evaluation_interval: How often, in seconds, to run the query.
Valid
                                    values are 10, 20, 30, and any multiple of
60, up
                                    to 3600.
        :param pending_period: How long, in seconds, a contributor must breach
                               continuously before it moves to ALARM.
```

```
        :param recovery_period: How long, in seconds, a contributor must stop
breaching
                                before it moves back to OK.
        :param description: The description of the alarm.
        :param alarm_actions: A list of ARNs to notify when the alarm fires, such
as an
                              Amazon SNS topic.
        """
        promql_criteria = {
            "Query": query,
            "PendingPeriod": pending_period,
            "RecoveryPeriod": recovery_period,
        }
        kwargs = {
            "AlarmName": alarm_name,
            "EvaluationCriteria": {"PromQLCriteria": promql_criteria},
            "EvaluationInterval": evaluation_interval,
        }
        if description is not None:
            kwargs["AlarmDescription"] = description
        if alarm_actions is not None:
            kwargs["AlarmActions"] = alarm_actions

        try:
            self.cloudwatch_client.put_metric_alarm(**kwargs)
            logger.info("Created PromQL alarm %s for query %s.", alarm_name,
query)
        except ClientError:
            logger.exception("Couldn't create PromQL alarm %s.", alarm_name)
            raise


    def describe_alarm_contributors(self, alarm_name):
        """
        Gets the contributors for a PromQL alarm. Each contributor is one series
that
        the alarm's query matched, identified by its label set. This is how you
find out
        *which* hosts, services, or pods are breaching, rather than only that
something
        is.

        :param alarm_name: The name of the PromQL alarm.
```

```
        :return: The list of contributors. Each contributor has a ContributorId,
a
                 ContributorAttributes map of the labels that identify the
series, a
                 StateReason, and the time it last changed state.
        """
        contributors = []
        try:
            next_token = None
            while True:
                kwargs = {"AlarmName": alarm_name}
                if next_token is not None:
                    kwargs["NextToken"] = next_token
                response =
self.cloudwatch_client.describe_alarm_contributors(**kwargs)
                contributors.extend(response["AlarmContributors"])
                next_token = response.get("NextToken")
                if not next_token:
                    break
        except ClientError:
            logger.exception("Couldn't get contributors for alarm %s.",
alarm_name)
            raise
        else:
            logger.info(
                "Got %s contributors for alarm %s.", len(contributors),
alarm_name
            )
            return contributors


    def put_alarm_mute_rule(
        self,
        name,
        expression,
        duration,
        alarm_names=None,
        timezone=None,
        description=None,
    ):
        """
        Creates or updates an alarm mute rule. While a mute rule is active the
targeted
        alarms keep evaluating and keep transitioning between states, but their
```

```
        configured actions do not fire. This is the supported way to suppress
        notifications during a known maintenance window instead of disabling
 alarm
        actions and hoping someone remembers to turn them back on.

        :param name: The name of the mute rule.
        :param expression: When the rule activates. For a recurring window, use a
                           five-field cron expression,
                           'cron(Minutes Hours Day-of-month Month Day-of-week)',
 such as
                           'cron(0 2 * * SUN)' for every Sunday at 2:00 AM. Note
 that
                           this is five fields, not the six that Amazon
 EventBridge
                           uses. For a one-time window, use 'at(yyyy-MM-
ddThh:mm)',
                           such as 'at(2026-09-05T02:00)'.
        :param duration: How long the mute window lasts once it activates, in ISO
 8601
                         duration format, from 'PT1M' (one minute) to 'P15D' (15
 days).
                         For example, 'PT2H' is two hours and 'P2DT12H' is two
 days and
                         12 hours.
        :param alarm_names: The names of up to 100 alarms to mute. If omitted,
 the rule
                            applies to all alarms in the account.
        :param timezone: The time zone the expression is evaluated in, such as
                         'America/Los_Angeles'.
        :param description: The description of the mute rule.
        """
        schedule = {"Expression": expression, "Duration": duration}
        if timezone is not None:
            schedule["Timezone"] = timezone

        kwargs = {"Name": name, "Rule": {"Schedule": schedule}}
        if alarm_names is not None:
            kwargs["MuteTargets"] = {"AlarmNames": alarm_names}
        if description is not None:
            kwargs["Description"] = description

        try:
            self.cloudwatch_client.put_alarm_mute_rule(**kwargs)
            logger.info("Put alarm mute rule %s.", name)
```

```
        except ClientError:
            logger.exception("Couldn't put alarm mute rule %s.", name)
            raise


    def get_alarm_mute_rule(self, name):
        """
        Gets the full configuration of an alarm mute rule, including its
schedule, the
        alarms it targets, and whether it is currently SCHEDULED, ACTIVE, or
EXPIRED.

        :param name: The name of the mute rule.
        :return: The mute rule.
        """
        try:
            response = self.cloudwatch_client.get_alarm_mute_rule(
                AlarmMuteRuleName=name
            )
        except ClientError:
            logger.exception("Couldn't get alarm mute rule %s.", name)
            raise
        else:
            logger.info("Got alarm mute rule %s.", name)
            return response


    def list_alarm_mute_rules(self, alarm_name=None, statuses=None):
        """
        Lists alarm mute rules in the account.

        :param alarm_name: When specified, only rules that target this alarm are
                           returned.
        :param statuses: When specified, only rules in these statuses are
returned.
                         Valid values are 'SCHEDULED', 'ACTIVE', and 'EXPIRED'.
        :return: The list of mute rule summaries.
        """
        summaries = []
        try:
            next_token = None
            while True:
                kwargs = {}
                if alarm_name is not None:
```

```
                    kwargs["AlarmName"] = alarm_name
                if statuses is not None:
                    kwargs["Statuses"] = statuses
                if next_token is not None:
                    kwargs["NextToken"] = next_token
                response = self.cloudwatch_client.list_alarm_mute_rules(**kwargs)
                summaries.extend(response.get("AlarmMuteRuleSummaries", []))
                next_token = response.get("NextToken")
                if not next_token:
                    break
        except ClientError:
            logger.exception("Couldn't list alarm mute rules.")
            raise
        else:
            logger.info("Got %s alarm mute rules.", len(summaries))
            return summaries


    def delete_alarm_mute_rule(self, name):
        """
        Deletes an alarm mute rule.

        :param name: The name of the mute rule.
        """
        try:
            self.cloudwatch_client.delete_alarm_mute_rule(AlarmMuteRuleName=name)
            logger.info("Deleted alarm mute rule %s.", name)
        except ClientError:
            logger.exception("Couldn't delete alarm mute rule %s.", name)
            raise


    def stop_otel_enrichment(self):
        """
        Turns off OTel enrichment for the account. Existing PromQL alarms are not
        deleted, but vended metrics stop being enriched with resource ARN and tag
        labels, so queries that select on those labels stop matching.
        """
        try:
            self.cloudwatch_client.stop_o_tel_enrichment()
            logger.info("Stopped OTel enrichment for this account.")
        except ClientError:
            logger.exception("Couldn't stop OTel enrichment.")
            raise
```

```
    def delete_alarms(self, alarm_names):
        """
        Deletes the specified alarms.

        :param alarm_names: The names of the alarms to delete.
        """
        try:
            self.cloudwatch_client.delete_alarms(AlarmNames=alarm_names)
            logger.info("Deleted alarms %s.", ", ".join(alarm_names))
        except ClientError:
            logger.exception("Couldn't delete alarms %s.", ",
 ".join(alarm_names))
            raise
```

Use the wrapper class to alarm on OpenTelemetry metrics with a PromQL query, inspect the contributors to the alarm, and mute it.

```
def usage_demo():
    """
    Walks through the OpenTelemetry metrics workflow in CloudWatch: turn on
    enrichment, alarm on a PromQL query, inspect the contributors that matched,
 mute
    the alarm for a maintenance window, then clean up.

    This scenario assumes OpenTelemetry metrics are already flowing into the
 account,
    either from an OpenTelemetry collector, the CloudWatch agent, or the ADOT
 SDK.
    """
    print("-" * 88)
    print("Welcome to the Amazon CloudWatch OpenTelemetry metrics demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    cw = CloudWatchOTelWrapper.from_client()

    alarm_name = "doc-example-promql-high-cpu"
    mute_rule_name = "doc-example-maintenance-window"
```

```
   print("Checking whether OTel enrichment is on for this account.")
   status = cw.get_otel_enrichment_status()
   started_enrichment_here = False
   if status == "Stopped":
       print("Enrichment is stopped. Starting it so vended metrics accept
PromQL.")
       cw.start_otel_enrichment()
       started_enrichment_here = True
   else:
       print("Enrichment is already running. Leaving it alone.")

   query = 'avg by (host_name)
(cpu_utilization_percent{service_name="checkout"}) > 80'
   print(f"\nCreating a PromQL alarm on: {query}")
   cw.create_promql_alarm(
       alarm_name,
       query,
       evaluation_interval=30,
       pending_period=300,
       recovery_period=120,
       description="Average CPU over 80% per host for the checkout service.",
   )
   print(
       "The alarm evaluates every 30 seconds. A host moves to ALARM after
breaching "
       "for 300 seconds straight, and back to OK after 120 seconds clean."
   )

   print("\nWaiting a moment for the first evaluation, then listing
contributors.")
   time.sleep(30)
   contributors = cw.describe_alarm_contributors(alarm_name)
   if not contributors:
       print(
           "No contributors yet. The query matched no series, which usually
means "
           "no OTel metrics with these labels have arrived. Send some OTel
metrics "
           "through the OTLP endpoint and run this again."
       )
   for contributor in contributors:
       labels = ", ".join(
           f"{key}={value}"
```

```
            for key, value in
 sorted(contributor["ContributorAttributes"].items())
        )
        print(f"  {contributor['ContributorId']}: {labels}")
        print(f"    reason: {contributor['StateReason']}")

    print(f"\nMuting {alarm_name} for a weekly two-hour maintenance window.")
    cw.put_alarm_mute_rule(
        mute_rule_name,
        expression="cron(0 2 * * SUN)",
        duration="PT2H",
        alarm_names=[alarm_name],
        timezone="America/Los_Angeles",
        description="Suppress checkout CPU pages during Sunday patching.",
    )
    rule = cw.get_alarm_mute_rule(mute_rule_name)
    print(f"Mute rule status is {rule.get('Status')}.")
    print(
        "While the window is active the alarm keeps evaluating and still changes
 "
        "state; only its actions are suppressed."
    )

    print(f"\nMute rules targeting {alarm_name}:")
    for summary in cw.list_alarm_mute_rules(alarm_name=alarm_name):
        print(f"  {summary.get('AlarmMuteRuleArn')} ({summary.get('Status')})")

    print("\nCleaning up.")
    cw.delete_alarm_mute_rule(mute_rule_name)
    cw.delete_alarms([alarm_name])
    if started_enrichment_here:
        print("Stopping OTel enrichment, since this demo started it.")
        cw.stop_otel_enrichment()

    print("\nThanks for watching!")
    print("-" * 88)
```

Configure an OpenTelemetry Collector to send the OTLP metrics that this example alarms on to the CloudWatch metrics endpoint. Metric ingestion over OTLP is not an AWS SDK operation, so this half of the example is collector configuration rather than SDK code.

```
# Purpose
#
# An OpenTelemetry Collector configuration that sends OTLP metrics to the Amazon
# CloudWatch metrics endpoint. Once metrics land in CloudWatch they are queryable
 with
# PromQL in Query Studio, and you can alarm on them with the PromQL alarm
 operations
# shown in cloudwatch_otel.py.
#
# Metric ingestion over OTLP is deliberately NOT an AWS SDK operation. There is
 no
# boto3 call that sends OTLP metrics. You send them with one of the following, in
# rough order of how integrated the CloudWatch experience is:
#
#   1. The CloudWatch agent (recommended). An AWS-managed OpenTelemetry Collector
 with
#      CloudWatch components pre-built. Adds entity correlation, runtime metrics,
 and
#      Container Insights support.
#   2. An upstream OpenTelemetry Collector. What this file configures.
#   3. A custom OpenTelemetry Collector build.
#   4. The AWS Distro for OpenTelemetry (ADOT) SDK, with no collector at all.
#
# Prerequisites
#
# * An OpenTelemetry Collector release (version 0.47.0 or later) that includes
 the
#   sigv4authextension. The contrib distribution does. See
#   https://github.com/open-telemetry/opentelemetry-collector-releases/releases
# * AWS credentials the collector can resolve. On Amazon EC2, attach the
#   CloudWatchAgentServerPolicy managed policy to the instance role. On Amazon
 EKS,
#   bind that policy to the collector's service account with IRSA. On premises,
 run
#   `aws configure` for an IAM user that has the same policy.
#
# Run the collector with:
#
#   otelcol-contrib --config otlp_collector_config.yaml
#
# Then point your instrumented application at the collector's OTLP HTTP endpoint
 on
```

```
# port 4318.
#
# For more information, see
# https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-
OTLPEndpoint.html

receivers:
  # The CloudWatch OTLP endpoints are HTTP 1.1 only and do not support gRPC. You
 can
  # still accept gRPC from your applications here and let the collector
 translate, but
  # this example keeps the receiver HTTP-only to mirror what is sent upstream.
  otlp:
    protocols:
      http:
        endpoint: "0.0.0.0:4318"

processors:
  # Batch to stay inside the endpoint's per-request limits: 1 MB uncompressed and
  # 1,000 datapoints, counted as the sum across ResourceMetrics, ScopeMetrics,
 and
  # Metrics. A batch of 200 leaves comfortable headroom.
  batch:
    send_batch_size: 200
    timeout: 10s

exporters:
  otlphttp:
    tls:
      insecure: false
    # Pattern: https://monitoring.{region}.amazonaws.com/v1/metrics
    endpoint: "https://monitoring.us-east-1.amazonaws.com/v1/metrics"
    # Only gzip and none are supported.
    compression: gzip
    auth:
      authenticator: sigv4auth

extensions:
  # SigV4 is the recommended authentication method, and the only one supported
 for
  # traces. For hosts outside AWS you can instead use the bearertokenauth
 extension
  # against the metrics or logs endpoints; see
```

```
  # https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-
OTLP-MetricsBearerTokenAuth.html
  sigv4auth:
    # The metrics endpoint signs as "monitoring". The logs endpoint signs as
 "logs"
    # and the traces endpoint as "xray".
    service: "monitoring"
    region: "us-east-1"

service:
  extensions: [sigv4auth]
  pipelines:
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlphttp]

# Endpoint limits worth designing around, all per account and Region:
#
#   Maximum TPS                  500
#   New series creation rate     1,000,000 per 10-minute window
#   Maximum request size         1 MB uncompressed
#   Maximum datapoint count      1,000 per request
#   Maximum metadata size        40 KB of labels and values per series per
 datapoint
#   Maximum label count          150 across Resource, Scope, and Datapoint
 attributes
#   Timestamp window             at most 10 minutes in the future, 14 days in the
 past
#
# Exceeding the TPS or new-series limits returns 429. The size, count, metadata,
# label, and timestamp limits return 400, and a request whose metrics are only
# partially invalid returns 200 with the valid metrics ingested.
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - DeleteAlarmMuteRule
  - DeleteAlarms
  - DescribeAlarmContributors
  - GetAlarmMuteRule
  - GetOTelEnrichment

- ListAlarmMuteRules
- PutAlarmMuteRule
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Define wrapper code that calls the CloudWatch OpenTelemetry operations.

```
# Turns on OTel enrichment for the account. Once enrichment is running,
 CloudWatch vended
# metrics that carry a resource identifier dimension, such as the Amazon EC2
# CPUUtilization metric with its InstanceId dimension, are decorated with
 resource ARN
# and resource tag labels and become queryable with PromQL.
#
# Resource tags on telemetry must already be enabled for the account before you
 call
# this operation.
#
# Note that the Ruby SDK renders the OTel prefix as +o_tel+, so the method is
# +start_o_tel_enrichment+ rather than +start_otel_enrichment+.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Boolean] true if enrichment was started; otherwise, false.
def otel_enrichment_started?(cloudwatch_client)
  cloudwatch_client.start_o_tel_enrichment
  true
rescue StandardError => e
  puts "Error starting OTel enrichment: #{e.message}"
```

```
  false
end

# Gets the current OTel enrichment status for the account.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [String, nil] 'Running' or 'Stopped', or nil if the status could not be
 read.
def otel_enrichment_status(cloudwatch_client)
  cloudwatch_client.get_o_tel_enrichment.status
rescue StandardError => e
  puts "Error getting OTel enrichment status: #{e.message}"
  nil
end

# Creates or updates an alarm that evaluates a PromQL query.
#
# A PromQL alarm differs from a classic metric alarm in a few ways. The query can
 match
# many series at once, and each matching series is tracked separately as a
 contributor.
# Instead of counting breaching periods, you specify durations: a contributor
 moves to
# ALARM after it breaches continuously for the pending period, and back to OK
 after it
# stops breaching for the recovery period. A PromQL alarm starts in the OK state
 rather
# than INSUFFICIENT_DATA.
#
# The +evaluation_criteria+ union is mutually exclusive with the classic
 +metric_name+
# and +metrics+ parameters. When you use it you must also set
 +evaluation_interval+, and
# you must not set +period+, +statistic+, +threshold+, +comparison_operator+,
# +evaluation_periods+, +datapoints_to_alarm+, or +treat_missing_data+.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the alarm, unique within the Region.
# @param criteria [Hash] The PromQL criteria, mirroring the +prom_ql_criteria+
 shape:
#   * +:query+ [String] The PromQL query to evaluate, such as
```

```
#     'avg(cpu_utilization_percent) > 80'. The comparison belongs in the query
 itself;
#     there is no separate threshold parameter.
#   * +:pending_period+ [Integer] How long, in seconds, a contributor must breach
#     continuously before it moves to ALARM.
#   * +:recovery_period+ [Integer] How long, in seconds, a contributor must stop
#     breaching before it moves back to OK.
# @param evaluation_interval [Integer] How often, in seconds, to run the query.
 Valid
#   values are 10, 20, 30, and any multiple of 60, up to 3600.
# @param alarm_description [String] A description of the alarm.
# @return [Boolean] true if the alarm was created or updated; otherwise, false.
def promql_alarm_created_or_updated?(
  cloudwatch_client,
  alarm_name,
  criteria,
  evaluation_interval,
  alarm_description
)
  cloudwatch_client.put_metric_alarm(
    alarm_name: alarm_name,
    alarm_description: alarm_description,
    evaluation_criteria: {
      prom_ql_criteria: {
        query: criteria[:query],
        pending_period: criteria[:pending_period],
        recovery_period: criteria[:recovery_period]
      }
    },
    evaluation_interval: evaluation_interval
  )
  true
rescue StandardError => e
  puts "Error creating PromQL alarm: #{e.message}"
  false
end

# Gets the contributors for a PromQL alarm. Each contributor is one series that
 the
# alarm's query matched, identified by its label set. This is how you find out
 which
# hosts, services, or pods are breaching, rather than only that something is.
#
```

```
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the PromQL alarm.
# @return [Array] The contributors, as Aws::CloudWatch::Types::AlarmContributor.
def alarm_contributors(cloudwatch_client, alarm_name)
  contributors = []
  next_token = nil

  loop do
    response = cloudwatch_client.describe_alarm_contributors(
      alarm_name: alarm_name,
      next_token: next_token
    )
    contributors.concat(response.alarm_contributors)
    next_token = response.next_token
    break if next_token.nil? || next_token.empty?
  end

  contributors
rescue StandardError => e
  puts "Error getting alarm contributors: #{e.message}"
  []
end

# Creates or updates an alarm mute rule. While a mute rule is active the targeted
 alarms
# keep evaluating and keep transitioning between states, but their configured
 actions do
# not fire. This is the supported way to suppress notifications during a known
# maintenance window, instead of disabling alarm actions and relying on someone
 to turn
# them back on.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param name [String] The name of the mute rule.
# @param schedule [Hash] The mute window, mirroring the +rule.schedule+ shape:
#   * +:expression+ [String] When the rule activates. For a recurring window, use
 a
#     five-field cron expression,
#     'cron(Minutes Hours Day-of-month Month Day-of-week)', such as
#     'cron(0 2 * * SUN)' for every Sunday at 2:00 AM. Note that this is five
 fields,
#     not the six that Amazon EventBridge uses. For a one-time window, use
```

```
#     'at(yyyy-MM-ddThh:mm)', such as 'at(2026-09-05T02:00)'.
#   * +:duration+ [String] How long the mute window lasts once it activates, in
 ISO 8601
#     duration format, from 'PT1M' (one minute) to 'P15D' (15 days). For example,
#     'PT2H' is two hours and 'P2DT12H' is two days and 12 hours.
#   * +:timezone+ [String] The time zone the expression is evaluated in, such as
#     'America/Los_Angeles'.
# @param alarm_names [Array] The names of up to 100 alarms to mute. If empty, the
 rule
#   applies to all alarms in the account.
# @param description [String] A description of the mute rule.
# @return [Boolean] true if the mute rule was created or updated; otherwise,
 false.
def alarm_mute_rule_created_or_updated?(
  cloudwatch_client,
  name,
  schedule,
  alarm_names,
  description
)
  params = {
    name: name,
    description: description,
    rule: {
      schedule: {
        expression: schedule[:expression],
        duration: schedule[:duration],
        timezone: schedule[:timezone]
      }
    }
  }
  params[:mute_targets] = { alarm_names: alarm_names } unless alarm_names.empty?

  cloudwatch_client.put_alarm_mute_rule(params)
  true
rescue StandardError => e
  puts "Error putting alarm mute rule: #{e.message}"
  false
end

# Gets the full configuration of an alarm mute rule, including its schedule, the
 alarms
# it targets, and whether it is currently SCHEDULED, ACTIVE, or EXPIRED.
#
```

```
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param name [String] The name of the mute rule.
# @return [Aws::CloudWatch::Types::GetAlarmMuteRuleOutput, nil] The mute rule, or
 nil on
#   error.
def alarm_mute_rule(cloudwatch_client, name)
  cloudwatch_client.get_alarm_mute_rule(alarm_mute_rule_name: name)
rescue StandardError => e
  puts "Error getting alarm mute rule: #{e.message}"
  nil
end

# Lists the alarm mute rules in the account, optionally filtered to the rules
 that
# target one alarm.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String, nil] When given, only rules that target this alarm
 are
#   returned.
# @return [Array] The mute rule summaries, as
#   Aws::CloudWatch::Types::AlarmMuteRuleSummary.
def alarm_mute_rules(cloudwatch_client, alarm_name = nil)
  summaries = []
  next_token = nil

  loop do
    response = cloudwatch_client.list_alarm_mute_rules(
      alarm_name: alarm_name,
      next_token: next_token
    )
    summaries.concat(response.alarm_mute_rule_summaries)
    next_token = response.next_token
    break if next_token.nil? || next_token.empty?
  end

  summaries
rescue StandardError => e
  puts "Error listing alarm mute rules: #{e.message}"
  []
end
```

```
# Deletes an alarm mute rule. The alarms it targeted resume firing their actions.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param name [String] The name of the mute rule.
# @return [Boolean] true if the mute rule was deleted; otherwise, false.
def alarm_mute_rule_deleted?(cloudwatch_client, name)
  cloudwatch_client.delete_alarm_mute_rule(alarm_mute_rule_name: name)
  true
rescue StandardError => e
  puts "Error deleting alarm mute rule: #{e.message}"
  false
end

# Turns off OTel enrichment for the account. Existing PromQL alarms are not
 deleted, but
# vended metrics stop being enriched with resource ARN and tag labels, so queries
 that
# select on those labels stop matching.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @return [Boolean] true if enrichment was stopped; otherwise, false.
def otel_enrichment_stopped?(cloudwatch_client)
  cloudwatch_client.stop_o_tel_enrichment
  true
rescue StandardError => e
  puts "Error stopping OTel enrichment: #{e.message}"
  false
end
```

Alarm on OpenTelemetry metrics with a PromQL query, inspect the contributors to the alarm, and mute it.

```
# Turns on OpenTelemetry enrichment if the account doesn't already have it on.
 Enrichment
# is an account-wide setting, so an example should only turn it off again if it
 was the
# one that turned it on.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
```

```
# @return [Boolean] true if this call started enrichment; otherwise, false.
def enrichment_started_by_example?(cloudwatch_client)
  puts 'Checking whether OTel enrichment is on for this account.'
  status = otel_enrichment_status(cloudwatch_client)
  if status == 'Stopped'
    puts 'Enrichment is stopped. Starting it so vended metrics accept PromQL.'
    return otel_enrichment_started?(cloudwatch_client)
  end

  puts "Enrichment status is '#{status}'. Leaving it alone."
  false
end

# Prints the contributors to a PromQL alarm, one line per matched series.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the PromQL alarm.
def report_alarm_contributors(cloudwatch_client, alarm_name)
  puts "\nContributors for '#{alarm_name}':"
  contributors = alarm_contributors(cloudwatch_client, alarm_name)
  if contributors.empty?
    puts '  None yet. The query matched no series, which usually means no OTel
 metrics ' \
         'with these labels have arrived.'
    return
  end

  contributors.each do |contributor|
    labels = contributor.contributor_attributes.sort.map { |k, v|
 "#{k}=#{v}" }.join(', ')
    puts "  #{contributor.contributor_id}: #{labels}"
    puts "    reason: #{contributor.state_reason}"
  end
end

# Mutes an alarm for a recurring weekly maintenance window.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param mute_rule_name [String] The name of the mute rule to create.
# @param alarm_name [String] The name of the alarm to mute.
def mute_alarm_for_maintenance(cloudwatch_client, mute_rule_name, alarm_name)
  puts "\nMuting '#{alarm_name}' for a weekly two-hour maintenance window."
```

```
  schedule = {
    expression: 'cron(0 2 * * SUN)',
    duration: 'PT2H',
    timezone: 'America/Los_Angeles'
  }
  return unless alarm_mute_rule_created_or_updated?(
    cloudwatch_client,
    mute_rule_name,
    schedule,
    [alarm_name],
    'Suppress checkout CPU pages during Sunday patching.'
  )

  rule = alarm_mute_rule(cloudwatch_client, mute_rule_name)
  puts "Mute rule status is #{rule.status}." unless rule.nil?
  puts 'While the window is active the alarm keeps evaluating and still changes '
 \
       'state; only its actions are suppressed.'
end

# Removes the mute rule and the alarm, and stops enrichment if this example
 started it.
#
# @param cloudwatch_client [Aws::CloudWatch::Client] An initialized CloudWatch
 client.
# @param alarm_name [String] The name of the alarm to delete.
# @param mute_rule_name [String] The name of the mute rule to delete.
# @param started_here [Boolean] Whether this example started OTel enrichment.
def clean_up(cloudwatch_client, alarm_name, mute_rule_name, started_here)
  puts "\nCleaning up."
  alarm_mute_rule_deleted?(cloudwatch_client, mute_rule_name)
  cloudwatch_client.delete_alarms(alarm_names: [alarm_name])
  return unless started_here

  puts 'Stopping OTel enrichment, since this example started it.'
  otel_enrichment_stopped?(cloudwatch_client)
end

# Walks through the OpenTelemetry metrics workflow in CloudWatch: turn on
 enrichment,
# alarm on a PromQL query, inspect the contributors that matched, mute the alarm
 for a
# maintenance window, then clean up.
#
```

```
# This scenario assumes OpenTelemetry metrics are already flowing into the
 account,
# either from an OpenTelemetry collector, the CloudWatch agent, or the ADOT SDK.
def run_me
  alarm_name = 'doc-example-promql-high-cpu'
  mute_rule_name = 'doc-example-maintenance-window'
  query = 'avg by (host_name) (cpu_utilization_percent{service_name="checkout"})
 > 80'
  # Replace us-east-1 with the AWS Region you're using for Amazon CloudWatch.
  region = 'us-east-1'

  cloudwatch_client = Aws::CloudWatch::Client.new(region: region)
  started_here = enrichment_started_by_example?(cloudwatch_client)

  puts "\nCreating a PromQL alarm on: #{query}"
  criteria = { query: query, pending_period: 300, recovery_period: 120 }
  unless promql_alarm_created_or_updated?(
    cloudwatch_client,
    alarm_name,
    criteria,
    30,
    'Average CPU over 80% per host for the checkout service.'
  )
    puts "Could not create alarm '#{alarm_name}'. Stopping."
    return
  end
  puts 'The alarm evaluates every 30 seconds. A host moves to ALARM after
 breaching ' \
       'for 300 seconds straight, and back to OK after 120 seconds clean.'

  report_alarm_contributors(cloudwatch_client, alarm_name)
  mute_alarm_for_maintenance(cloudwatch_client, mute_rule_name, alarm_name)

  puts "\nMute rules targeting '#{alarm_name}':"
  alarm_mute_rules(cloudwatch_client, alarm_name).each do |summary|
    puts "  #{summary.alarm_mute_rule_arn} (#{summary.status})"
  end

  clean_up(cloudwatch_client, alarm_name, mute_rule_name, started_here)
end
```

- For API details, see the following topics in *AWS SDK for Ruby API Reference*.

- DeleteAlarmMuteRule
- DeleteAlarms
- DescribeAlarmContributors
- GetAlarmMuteRule
- GetOTelEnrichment
- ListAlarmMuteRules
- PutAlarmMuteRule
- PutMetricAlarm
- StartOTelEnrichment
- StopOTelEnrichment

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Using property variables in monitoring dashboards to monitor multiple serverless functions

The following code example shows how to:

- Create Lambda functions for monitoring
- Create a CloudWatch dashboard
- Add a property variable to the dashboard
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# CloudWatch Dashboard with Lambda Function Variable Script
# This script creates a CloudWatch dashboard with a property variable for Lambda
 function names

set -euo pipefail

# Security: Set restrictive umask
umask 0077

# Set up logging with secure permissions
LOG_FILE="cloudwatch-dashboard-script-v4.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
echo "Starting script execution at $(date)" >> "$LOG_FILE"

# Function to log commands and their output (with sensitive data sanitization)
log_cmd() {
    local cmd="$1"
    local sanitized_cmd="${cmd//--password*/--password [REDACTED]}"
    sanitized_cmd="${sanitized_cmd//--secret*/--secret [REDACTED]}"
    echo "$(date): Running command: $sanitized_cmd" >> "$LOG_FILE"
    eval "$cmd" 2>&1 | tee -a "$LOG_FILE"
    return ${PIPESTATUS[0]}
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"

    if [ $cmd_status -ne 0 ] || echo "$cmd_output" | grep -qi "error"; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        # Sanitize output before logging
        local sanitized_output="${cmd_output//arn:aws:iam::[0-9]*/
arn:aws:iam::ACCOUNT_ID}"
        echo "Command output: $sanitized_output" | tee -a "$LOG_FILE"
        cleanup_resources
        exit 1
    fi
}
```

```
# Trap errors and cleanup
trap 'cleanup_resources' EXIT ERR INT TERM

# Function to clean up resources
cleanup_resources() {
    local exit_code=$?

    echo "" | tee -a "$LOG_FILE"
    echo "==========================================" | tee -a "$LOG_FILE"
    echo "CLEANUP PROCESS" | tee -a "$LOG_FILE"
    echo "==========================================" | tee -a "$LOG_FILE"

    if [ -n "${DASHBOARD_NAME:-}" ]; then
        echo "Deleting CloudWatch dashboard: $DASHBOARD_NAME" | tee -a
 "$LOG_FILE"
        aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" 2>&1
 >> "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION1:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION1" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION2:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION2" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole 2>&1 >>
 "$LOG_FILE" || true

        echo "Deleting IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam delete-role --role-name "$ROLE_NAME" 2>&1 >> "$LOG_FILE" || true
    fi

    # Clean up temporary files securely
```

```
    shred -vfz -n 3 trust-policy.json lambda_function.py lambda_function.zip 2>/
dev/null || rm -f trust-policy.json lambda_function.py lambda_function.zip

    echo "Cleanup completed." | tee -a "$LOG_FILE"

    return $exit_code
}

# Validate AWS CLI is installed and authenticated
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed" | tee -a "$LOG_FILE"
    exit 1
fi

if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS CLI is not properly authenticated" | tee -a "$LOG_FILE"
    exit 1
fi

# Get AWS region with validation
AWS_REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $AWS_REGION" | tee -a
 "$LOG_FILE"
else
    echo "Using AWS region: $AWS_REGION" | tee -a "$LOG_FILE"
fi

# Validate region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]$ ]]; then
    echo "ERROR: Invalid AWS region format: $AWS_REGION" | tee -a "$LOG_FILE"
    exit 1
fi

# Generate unique identifiers using secure random with validation
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ] || [ ${#RANDOM_ID} -ne 12 ]; then
    echo "ERROR: Failed to generate valid random identifier" | tee -a "$LOG_FILE"
    exit 1
fi

DASHBOARD_NAME="LambdaMetricsDashboard-${RANDOM_ID}"
LAMBDA_FUNCTION1="TestFunction1-${RANDOM_ID}"
```

```
LAMBDA_FUNCTION2="TestFunction2-${RANDOM_ID}"
ROLE_NAME="LambdaExecutionRole-${RANDOM_ID}"

# Validate resource names don't exceed AWS limits
if [ ${#DASHBOARD_NAME} -gt 128 ] || [ ${#LAMBDA_FUNCTION1} -gt 64 ] ||
 [ ${#ROLE_NAME} -gt 64 ]; then
    echo "ERROR: Generated resource names exceed AWS limits" | tee -a "$LOG_FILE"
    exit 1
fi

echo "Using random identifier: $RANDOM_ID" | tee -a "$LOG_FILE"
echo "Dashboard name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda function names: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "IAM role name: $ROLE_NAME" | tee -a "$LOG_FILE"

# Create IAM role for Lambda functions
echo "Creating IAM role for Lambda..." | tee -a "$LOG_FILE"
TRUST_POLICY='{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'

echo "$TRUST_POLICY" > trust-policy.json
chmod 600 trust-policy.json

# Validate JSON before use
if ! python3 -m json.tool trust-policy.json > /dev/null 2>&1; then
    echo "ERROR: Invalid trust policy JSON" | tee -a "$LOG_FILE"
    exit 1
fi

ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name '$ROLE_NAME' --assume-
role-policy-document file://trust-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=cloudwatch-streams --output json")
check_error "$ROLE_OUTPUT" $? "Failed to create IAM role"
```

```
ROLE_ARN=$(echo "$ROLE_OUTPUT" | python3 -c "import sys, json;
 print(json.load(sys.stdin)['Role']['Arn'])" 2>/dev/null)
if [ -z "$ROLE_ARN" ]; then
    echo "ERROR: Failed to extract Role ARN" | tee -a "$LOG_FILE"
    exit 1
fi
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Attach Lambda basic execution policy to the role
echo "Attaching Lambda execution policy to role..." | tee -a "$LOG_FILE"
POLICY_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name '$ROLE_NAME' --
policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole")
check_error "$POLICY_OUTPUT" $? "Failed to attach policy to role"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..." | tee -a "$LOG_FILE"
sleep 10

# Create simple Python Lambda function code with security validation
echo "Creating Lambda function code..." | tee -a "$LOG_FILE"
cat > lambda_function.py << 'LAMBDA_EOF'
def handler(event, context):
    print("Lambda function executed successfully")
    return {
        'statusCode': 200,
        'body': 'Success'
    }
LAMBDA_EOF

chmod 600 lambda_function.py

# Validate Python syntax
if ! python3 -m py_compile lambda_function.py 2>/dev/null; then
    echo "ERROR: Invalid Python syntax in Lambda function" | tee -a "$LOG_FILE"
    exit 1
fi

# Zip the Lambda function code
zip -j -q lambda_function.zip lambda_function.py
if [ ! -f lambda_function.zip ]; then
    echo "ERROR: Failed to create lambda_function.zip" | tee -a "$LOG_FILE"
    exit 1
fi
```

```
chmod 600 lambda_function.zip

# Validate zip file integrity
if ! unzip -t lambda_function.zip > /dev/null 2>&1; then
    echo "ERROR: Created zip file is corrupted" | tee -a "$LOG_FILE"
    exit 1
fi

# Create first Lambda function
echo "Creating first Lambda function: $LAMBDA_FUNCTION1..." | tee -a "$LOG_FILE"
LAMBDA1_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION1' --runtime python3.11 --role '$ROLE_ARN' --handler
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA1_OUTPUT" $? "Failed to create first Lambda function"

# Create second Lambda function
echo "Creating second Lambda function: $LAMBDA_FUNCTION2..." | tee -a "$LOG_FILE"
LAMBDA2_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION2' --runtime python3.11 --role '$ROLE_ARN' --handler
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA2_OUTPUT" $? "Failed to create second Lambda function"

# Invoke Lambda functions to generate some metrics
echo "Invoking Lambda functions to generate metrics..." | tee -a "$LOG_FILE"
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION1' --payload '{}' /
dev/null" || true
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION2' --payload '{}' /
dev/null" || true

# Create CloudWatch dashboard with property variable
echo "Creating CloudWatch dashboard with property variable..." | tee -a
 "$LOG_FILE"

# Create dashboard body with proper escaping and validation
DASHBOARD_BODY=$(cat <<EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
```

```
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "$LAMBDA_FUNCTION1" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$AWS_REGION",
        "title": "Lambda Invocations",
        "period": 300,
        "stat": "Sum"
      }
    }
  ]
}
EOF
)

# Validate JSON before sending
if ! echo "$DASHBOARD_BODY" | python3 -m json.tool > /dev/null 2>&1; then
    echo "ERROR: Dashboard body is not valid JSON" | tee -a "$LOG_FILE"
    exit 1
fi

# First create a basic dashboard without variables
echo "Creating initial dashboard without variables..." | tee -a "$LOG_FILE"
DASHBOARD_OUTPUT=$(aws cloudwatch put-dashboard --dashboard-name
 "$DASHBOARD_NAME" --dashboard-body "$DASHBOARD_BODY" --output json 2>&1)
check_error "$DASHBOARD_OUTPUT" $? "Failed to create initial CloudWatch
 dashboard"

# Now let's try to add a property variable using the console instructions
echo "To complete the tutorial, please follow these steps in the CloudWatch
 console:" | tee -a "$LOG_FILE"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/" | tee -a "$LOG_FILE"
echo "2. Navigate to Dashboards and select your dashboard: $DASHBOARD_NAME" | tee
 -a "$LOG_FILE"
echo "3. Choose Actions > Variables > Create a variable" | tee -a "$LOG_FILE"
echo "4. Choose Property variable" | tee -a "$LOG_FILE"
echo "5. For Property that the variable changes, choose FunctionName" | tee -a
 "$LOG_FILE"
echo "6. For Input type, choose Select menu (dropdown)" | tee -a "$LOG_FILE"
echo "7. Choose Use the results of a metric search" | tee -a "$LOG_FILE"
```

```
echo "8. Choose Pre-built queries > Lambda > Errors" | tee -a "$LOG_FILE"
echo "9. Choose By Function Name and then choose Search" | tee -a "$LOG_FILE"
echo "10. (Optional) Configure any secondary settings as desired" | tee -a
 "$LOG_FILE"
echo "11. Choose Add variable" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "The dashboard has been created and can be accessed at:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"

# Verify dashboard creation
echo "Verifying dashboard creation..." | tee -a "$LOG_FILE"
VERIFY_OUTPUT=$(aws cloudwatch get-dashboard --dashboard-name "$DASHBOARD_NAME"
 --output json 2>&1)
check_error "$VERIFY_OUTPUT" $? "Failed to verify dashboard creation"

echo "" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "DASHBOARD CREATED SUCCESSFULLY" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "Dashboard Name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda Functions: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "You can view your dashboard in the CloudWatch console:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"

# Auto-confirm cleanup
echo "" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "CLEANUP CONFIRMATION" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "The following resources were created:" | tee -a "$LOG_FILE"
echo "- CloudWatch Dashboard: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
echo "- IAM Role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "Auto-confirming cleanup of all created resources..." | tee -a "$LOG_FILE"
```

```
echo "Script completed successfully." | tee -a "$LOG_FILE"
exit 0
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards
  - DeleteFunction
  - DeleteRole
  - DetachRolePolicy
  - GetDashboard
  - Invoke
  - PutDashboard

For a complete list of AWS SDK developer guides and code examples, see Using CloudWatch with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Logging Amazon CloudWatch API and console operations with AWS CloudTrail

Amazon CloudWatch, CloudWatch Synthetics, CloudWatch RUM, Amazon Q Developer operational investigations, Network Flow Monitor, and Internet Monitor are integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls made by or on behalf of your AWS account. The captured calls include calls from the CloudWatch console and code calls to CloudWatch API operations. Using the information collected by CloudTrail, you can determine the request that was made to CloudWatch, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a

trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

Note

For information about CloudWatch Logs API calls that are logged in CloudTrail, see CloudWatch Logs information in CloudTrail.

Topics

- CloudWatch information in CloudTrail
- CloudWatch data events in CloudTrail
- Query generation information in CloudTrail
- Amazon Q Developer operational investigations events in CloudTrail
- Network Flow Monitor in CloudTrail
- Network Flow Monitor data plane events in CloudTrail

- Internet Monitor in CloudTrail
- CloudWatch Synthetics information in CloudTrail
- CloudWatch RUM information in CloudTrail
- CloudWatch RUM data plane events in CloudTrail
- Network Synthetic Monitor information in CloudTrail
- CloudWatch Observability Access Manager information in CloudTrail
- CloudWatch Observability Admin information in CloudTrail
- CloudWatch Application Signals information in CloudTrail
- CloudWatch Application Insights information in CloudTrail

CloudWatch information in CloudTrail

CloudWatch supports logging the following actions as events in CloudTrail log files:

- DeleteAlarmActions
- DeleteAnomalyDetector
- DeleteDashboards
- DeleteInsightRules
- DeleteMetricStream
- DescribeAlarmHistory
- DescribeAlarms
- DescribeAlarmsForMetric
- DescribeAnomalyDetectors
- DescribeInsightRules
- DisableAlarmActions
- DisableInsightRules
- EnableAlarmActions
- EnableInsightRules
- GetDashboard
- GetInsightRuleReport
- GetMetricStream

- ListDashboards
- ListManagedInsightRules
- ListMetricStreams
- ListTagsForResource
- PutAnomalyDetector
- PutCompositeAlarm
- PutDashboard
- PutInsightRule
- PutManagedInsightRules
- PutLogAlarm
- PutMetricAlarm
- PutMetricStream
- SetAlarmState
- StartMetricStreams
- StopMetricStreams
- TagResource
- UntagResource

Example: CloudWatch log file entries

The following example shows a CloudTrail log entry that demonstrates the `PutMetricAlarm` action.

```
{
    "Records": [{
        "eventVersion": "1.01",
        "userIdentity": {
            "type": "Root",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:root",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID"
        },
        "eventTime": "2014-03-23T21:50:34Z",
        "eventSource": "monitoring.amazonaws.com",
```

```
        "eventName": "PutMetricAlarm",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "threshold": 50.0,
            "period": 60,
            "metricName": "CloudTrail Test",
            "evaluationPeriods": 3,
            "comparisonOperator": "GreaterThanThreshold",
            "namespace": "AWS/CloudWatch",
            "alarmName": "CloudTrail Test Alarm",
            "statistic": "Sum"
        },
        "responseElements": null,
        "requestID": "29184022-b2d5-11e3-a63d-9b463e6d0ff0",
        "eventID": "b096d5b7-dcf2-4399-998b-5a53eca76a27"
    },
    ..additional entries
  ]
  }
```

The following log file entry shows that a user called the CloudWatch Events `PutRule` action.

```
{
         "eventVersion":"1.03",
         "userIdentity":{
            "type":"Root",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:root",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext":{
               "attributes":{
                  "mfaAuthenticated":"false",
                  "creationDate":"2015-11-17T23:56:15Z"
               }
            }
         },
         "eventTime":"2015-11-18T00:11:28Z",
         "eventSource":"events.amazonaws.com",
         "eventName":"PutRule",
         "awsRegion":"us-east-1",
```

```
         "sourceIPAddress":"AWS Internal",
         "userAgent":"AWS CloudWatch Console",
         "requestParameters":{
            "description":"",
            "name":"cttest2",
            "state":"ENABLED",
            "eventPattern":"{\"source\":[\"aws.ec2\"],\"detail-type\":[\"EC2 Instance
 State-change Notification\"]}",
            "scheduleExpression":""
         },
         "responseElements":{
            "ruleArn":"arn:aws:events:us-east-1:123456789012:rule/cttest2"
         },
         "requestID":"e9caf887-8d88-11e5-a331-3332aa445952",
         "eventID":"49d14f36-6450-44a5-a501-b0fdcdfaeb98",
         "eventType":"AwsApiCall",
         "apiVersion":"2015-10-07",
         "recipientAccountId":"123456789012"
}
```

The following log file entry shows that a user called the CloudWatch Logs `CreateExportTask` action.

```
{
        "eventVersion": "1.03",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/someuser",
            "accountId": "123456789012",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "someuser"
        },
        "eventTime": "2016-02-08T06:35:14Z",
        "eventSource": "logs.amazonaws.com",
        "eventName": "CreateExportTask",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "127.0.0.1",
        "userAgent": "aws-sdk-ruby2/2.0.0.rc4 ruby/1.9.3 x86_64-linux Seahorse/0.1.0",
        "requestParameters": {
            "destination": "yourdestination",
            "logGroupName": "yourloggroup",
            "to": 123456789012,
```

```
            "from": 0,
            "taskName": "yourtask"
        },
        "responseElements": {
            "taskId": "15e5e534-9548-44ab-a221-64d9d2b27b9b"
        },
        "requestID": "1cd74c1c-ce2e-12e6-99a9-8dbb26bd06c9",
        "eventID": "fd072859-bd7c-4865-9e76-8e364e89307c",
        "eventType": "AwsApiCall",
        "apiVersion": "20140328",
        "recipientAccountId": "123456789012"
}
```


CloudWatch data events in CloudTrail

CloudTrail captures API activities for two groups of CloudWatch data plane operations. The first group is the metric data-plane APIs: GetMetricData, GetMetricWidgetImage, PutMetricData, GetMetricStatistics, and ListMetrics. The second group is the Prometheus-compatible query APIs for Prometheus Query Language (PromQL): `QueryMetrics`, `GetSeries`, and `GetLabels`. For more information about these APIs, see Prometheus-compatible APIs.

CloudTrail logs each data event under one of two CloudWatch data resource types, as shown in the following table.

Data event	Data resource type
`GetMetricData`	`AWS::CloudWatch::Metric`
`GetMetricWidgetImage`	`AWS::CloudWatch::Metric`
`PutMetricData`	`AWS::CloudWatch::Metric`
`PutMetricData` (metrics sent through the OpenTelemetry Protocol (OTLP) endpoint)	`AWS::CloudWatch::Dataset`
`GetMetricStatistics`	`AWS::CloudWatch::Metric`
`ListMetrics`	`AWS::CloudWatch::Metric`
`QueryMetrics`	`AWS::CloudWatch::Dataset`

Data event	Data resource type
`GetSeries`	`AWS::CloudWatch::Dataset`
`GetLabels`	`AWS::CloudWatch::Dataset`

> **Note**
>
> CloudTrail records `QueryMetrics`, `GetSeries`, and `GetLabels` query requests with `readOnly` set to `true` and an `eventSource` of `monitoring.amazonaws.com`. To log PromQL query requests, enable data events for the CloudWatch dataset resource type.

> **Note**
>
> Metrics that you send to CloudWatch through the OpenTelemetry Protocol (OTLP) endpoint are recorded as `PutMetricData` data events under the `AWS::CloudWatch::Dataset` resource type (not `AWS::CloudWatch::Metric`). These events include an `additionalEventData` field with the value `{"protocol": "OTLP"}`. To log these OTLP metric ingestion events, enable data events for the CloudWatch dataset resource type.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

By default, CloudTrail doesn't log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the CloudWatch resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. For more information about how to log data events, see Logging data events with the AWS Management Console and Logging data events with the AWS Command Line Interface in the *AWS CloudTrail User Guide*.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify the API request, the IP address of the requester, the requester's identity, and the date and time of the request. Logging these data events with CloudTrail helps you enable operational and risk auditing, governance, and compliance of your AWS account.

> **Note**
>
> When you view the **GetMetricData** events in CloudTrail, you might see more calls than the calls that you initiated. This is because CloudWatch logs events to CloudTrail for **GetMetricData** actions that are initiated by internal components. For example, you'll see **GetMetricData** calls initiated by CloudWatch dashboards to refresh widget data, and **GetMetricData** calls initiated by a monitoring account to retrieve data from a source account, in cross-account observability. These internally-initiated calls don't incur CloudWatch charges, but they do count toward the number of events logged in CloudTrail, and CloudTrail charges according to the number of events logged.

The following is an example of a CloudTrail event for a **GetMetricData** operation.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDA2NYTR2EPCTNY7AF3L",
        "arn": "arn:aws:iam::111122223333:user/admin",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE1234567890",
        "userName": "admin"
    },
    "eventTime": "2024-05-08T16:20:34Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GetMetricData",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "99.45.3.7",
    "userAgent": "aws-cli/2.13.23 Python/3.11.5 Darwin/23.4.0 exe/x86_64 prompt/off
 command/cloudwatch.get-metric-data",
    "requestParameters": {
```

```
        "metricDataQueries": [{
                "id": "e1",
                "expression": "m1 / m2",
                "label": "ErrorRate"
            },
            {
                "id": "m1",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum",
                    "unit": "Count"
                },
                "returnData": false
            },
            {
                "id": "m2",
                "metricStat": {
                    "metric": {
                        "namespace": "CWAgent",
                        "metricName": "disk_used_percent",
                        "dimensions": [{
                            "name": "LoadBalancerName",
                            "value": "EXAMPLE4623a5cb6a7384c5229"
                        }]
                    },
                    "period": 300,
                    "stat": "Sum"
                },
                "returnData": true
            }
        ],
        "startTime": "Apr 19, 2024, 4:00:00 AM",
        "endTime": "May 8, 2024, 4:30:00 AM"
    },
    "responseElements": null,
    "requestID": "EXAMPLE-57ac-47d5-938c-f5917c6799d5",
```

```
    "eventID": "EXAMPLE-211c-404b-b13d-36d93c8b4fbf",
    "readOnly": true,
    "resources": [{
        "type": "AWS::CloudWatch::Metric"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
}
```

The following is an example of a CloudTrail event for a **PutMetricData** operation.

```
{
      "eventVersion": "1.11",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "111122223333:example.amazon.com",
        "arn": "arn:aws:sts::111122223333:assumed-role/
cloudwatch.full.access/example.amazon.com",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLE1234567890",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AROA3FLD4LJVPWYJ6BCNM",
            "arn": "arn:aws:iam::111122223333:role/cloudwatch.full.access",
            "accountId": "111122223333",
            "userName": "cloudwatch.full.access"
          },
          "attributes": {
            "creationDate": "2025-06-19T23:19:50Z",
            "mfaAuthenticated": "false"
          }
        }
      },
      "eventTime": "2025-06-19T23:51:04Z",
      "eventSource": "monitoring.amazonaws.com",
```

```
    "eventName": "PutMetricData",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
      "namespace": "CloudTrailTests",
      "metricData": [
        {
          "metricName": "CloudTrailPutMetricDataTest",
          "dimensions": [
            {
              "name": "TestDimName",
              "value": "TestDimValue"
            }
          ]
        }
      ]
    },
    "responseElements": null,
    "requestID": "877db913-2620-4929-97f3-f3c93c6f689b",
    "eventID": "0c0c4516-75f4-4b27-8a83-213821a96a2b",
    "readOnly": false,
    "resources": [
      {
        "type": "AWS::CloudWatch::Metric"
      }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
      "tlsVersion": "TLSv1.3",
      "cipherSuite": "TLS_AES_128_GCM_SHA256",
      "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
  }
```

The following is an example of a CloudTrail event for a **PutMetricData** operation for metrics that were sent using the OpenTelemetry Protocol (OTLP).

```
{
    "eventVersion": "1.11",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAEXAMPLEID:i-EXAMPLE0123456789",
        "arn": "arn:aws:sts::111122223333:assumed-role/CloudWatchAgentServerRole/i-
EXAMPLE0123456789",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAEXAMPLEID",
                "arn": "arn:aws:iam::111122223333:role/CloudWatchAgentServerRole",
                "accountId": "111122223333",
                "userName": "CloudWatchAgentServerRole"
            },
            "attributes": {
                "creationDate": "2026-10-19T11:11:36Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2026-10-19T12:40:24Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "PutMetricData",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "CWAgent/1.300073.0 (go1.26; linux; amd64) inputs:(otlp) aws-sdk-go-
v2/1.36.3",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "protocol": "OTLP"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": false,
    "resources": [
        {
            "type": "AWS::CloudWatch::Dataset",
            "ARN": "arn:aws:cloudwatch:us-east-1:111122223333:dataset/default"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
```

```
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
}
```

The following is an example of a CloudTrail event for a **QueryMetrics** (PromQL) operation.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/janedoe",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "janedoe"
    },
    "eventTime": "2026-06-05T19:37:25Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "QueryMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.7",
    "userAgent": "awscurl/0.36",
    "requestParameters": {
        "path": "/api/v1/query",
        "query": "sum({\"http.server.active_requests\"})"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "readOnly": true,
    "resources": [{
        "type": "AWS::CloudWatch::Dataset"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
```

```
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "monitoring.us-east-1.amazonaws.com"
    }
}
```

Query generation information in CloudTrail

CloudTrail logging for Query generator console events is also supported. Query generator is currently supported for CloudWatch Metric Insights and CloudWatch Logs Insights. In these CloudTrail events, the `eventSource` is `monitoring.amazonaws.com`.

The following example shows a CloudTrail log entry that demonstrates the **GenerateQuery** action in CloudWatch Metrics Insights.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2020-04-08T21:43:24Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "monitoring.amazonaws.com",
    "eventName": "GenerateQuery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
```

```
    "userAgent": "exampleUserAgent",
    "requestParameters": {
        "query_ask": "***",
        "query_type": "MetricsInsights",
        "metrics_insights": {
            "aws_namespaces": [
                "AWS/S3",
                "AWS/Lambda",
                "AWS/DynamoDB"
            ]
        },
        "include_description": true
    },
    "responseElements": null,
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Amazon Q Developer operational investigations events in CloudTrail

Amazon Q Developer operational investigations supports logging the following actions as events in CloudTrail log files.

- CreateInvestigationGroup
- GetInvestigationGroup
- DeleteInvestigationGroup
- ListInvestigationGroup
- PutInvestigationGroupPolicy
- DeleteInvestigationGroupPolicy
- ListTagsForResource
- GetInvestigationGroupPolicy
- TagResource

- UntagResource
- UpdateInvestigationGroup

Example: Amazon Q Developer operational investigations log file entries

The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `CreateInvestigationGroup` action.


```
{
 "eventVersion": "1.09",
 "userIdentity": {
  "type": "AssumedRole",
  "principalId": "EX_PRINCIPAL_ID",
  "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
  "accountId": "123456789012",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "sessionContext": {
   "sessionIssuer": {
    "type": "Role",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:role/role_name",
    "accountId": "123456789012",
    "userName": "SAMPLE_NAME"
   },
   "attributes": {
    "creationDate": "2024-10-30T18:42:05Z",
    "mfaAuthenticated": "false"
   }
  }
 },
 "eventTime": "2024-10-30T18:48:26Z",
 "eventSource": "aiops.amazonaws.com",
 "eventName": "CreateInvestigationGroup",
 "awsRegion": "us-east-1",
 "sourceIPAddress": "127.0.0.1",
 "userAgent": "exampleUserAgent",
 "requestParameters": {
  "name": "exampleName",
  "roleArn": "arn:aws:iam::123456789012:role/role_name" },
 "responseElements": {
```

```
   "arn": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
  },
  "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
  "requestId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "123456789012",
  "eventCategory": "Management"
}
```

The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `CreateInvestigationEvent` action.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
   "type": "AssumedRole",
   "principalId": "EX_PRINCIPAL_ID",
   "arn": "arn:aws:sts::123456789012:assumed-role/role_name",
   "accountId": "123456789012",
   "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
   "sessionContext": {
    "sessionIssuer": {
     "type": "Role",
     "principalId": "EX_PRINCIPAL_ID",
     "arn": "arn:aws:iam::123456789012:role/role_name",
     "accountId": "123456789012",
     "userName": "SAMPLE_NAME"
    },
    "attributes": {
     "creationDate": "2024-10-30T16:17:49Z",
     "mfaAuthenticated": "false"
    }
   }
  },
  "eventTime": "2024-10-30T16:35:34Z",
  "eventSource": "aiops.amazonaws.com",
  "eventName": "CreateInvestigationEvent",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "exampleUserAgent",
  "requestParameters": {
```

```
    "identifier": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
    "investigationId": "bcdef01234567890",
    "clientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "type": "METRIC_OBSERVATION",
    "body": "***"
   },
   "responseElements": {
    "investigationGroupArn": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
    "investigationId": "bcdef01234567890",
    "investigationEventId": "14567890abcdef0g"
   },
   "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
   "eventId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
   "readOnly": false,
   "resources": [{
    "accountId": "123456789012",
    "type": "AWS::AIOps::InvestigationGroup",
    "ARN": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
   }],
   "eventType": "AwsApiCall",
   "managementEvent": false,
   "recipientAccountId": "123456789012",
   "eventCategory": "Data"
 }
```

The following example shows a Amazon Q Developer operational investigations log entry that demonstrates the `UpdateInvestigationEvent` action.

```
{
 "eventVersion": "1.09",
 "userIdentity": {
  "type": "AssumedRole",
  "principalId": "EX_PRINCIPAL_ID",
  "arn": "arn:aws:sts::123456789012:assumed-role/role_name",
  "accountId": "123456789012",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "sessionContext": {
   "sessionIssuer": {
    "type": "Role",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:role/role_name",
```

```
    "accountId": "123456789012",
    "userName": "SAMPLE_NAME"
   },
   "attributes": {
    "creationDate": "2024-10-30T16:17:49Z",
    "mfaAuthenticated": "false"
   }
  }
 },
 "eventTime": "2024-10-30T16:24:48Z",
 "eventSource": "aiops.amazonaws.com",
 "eventName": "UpdateInvestigationEvent",
 "awsRegion": "us-east-1",
 "sourceIPAddress": "127.0.0.1",
 "userAgent": "exampleUserAgent",
 "requestParameters": {
  "identifier": "arn:aws:aiops:us-east-1:123456789012:investigation-
group/021345abcdef67890",
  "investigationId": "bcdef01234567890",
  "investigationEventId": "14567890abcdef0g",
  "comment": "***"
 },
 "responseElements": null,
 "requestId": "e9caf887-8d88-11e5-a331-3332aa445952",
 "eventId": "49d14f36-6450-44a5-a501-b0fdcdfaeb98",
 "readOnly": false,
 "resources": [{
  "accountId": "123456789012",
  "type": "AWS::AIOps::InvestigationGroup",
  "ARN": "arn:aws:aiops:us-east-1:123456789012:investigation-group/021345abcdef67890"
 }],
 "eventType": "AwsApiCall",
 "managementEvent": false,
 "recipientAccountId": "123456789012",
 "eventCategory": "Data"
}
```


Network Flow Monitor in CloudTrail

Network Flow Monitor supports logging the following actions as events in CloudTrail log files.

- CreateMonitor

- CreateScope
- DeleteMonitor
- DeleteScope
- GetMonitor
- GetScope
- ListMonitors
- ListScopes
- ListTagsForResource
- TagResource
- UntagResource
- UpdateMonitor
- UpdateScope

Example: Network Flow Monitor log file entry

The following example shows a Network Flow Monitor CloudTrail log file entry that demonstrates the `CreateMonitor` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId": "123456789012",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
```

```
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "networkflowmonitor.amazonaws.com",
    "eventName": "CreateMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
        "MonitorName": "TestMonitor",
        "ClientToken": "33551db7-1618-4aab-cdef-EXAMPLE33333",
        "LocalResources": [
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEbbbbb"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEccccc"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEddddd"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEeeeee"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEfffff"
            },
            {
                "Type": "AWS::EC2::Subnet",
                "Identifier": "subnet-cdef-EXAMPLEggggg"
            }
        ]
    },
    "responseElements": {
        "Access-Control-Expose-Headers": "*",
        "MonitorArn": "arn:aws:networkflowmonitor:us-east-1:000000000000:monitor/
TestMonitor",
```

```
        "MonitorName": "TestMonitor",
        "MonitorStatus": "ACTIVE"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "networkflowmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": null,
        "responseElements": null,
```

```
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```

Network Flow Monitor data plane events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch-NetworkFlowMonitor data plane operations.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

To enable logging of Network Flow Monitor data events in CloudTrail files, you'll need to enable the logging of data plane API activity in CloudTrail. See Logging data events for trails for more information.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch-NetworkFlowMonitor data plane APIs, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the data plane APIs using CloudTrail can help you with operational and risk auditing, governance, and compliance of your AWS account.

The following are data plane APIs in Network Flow Monitor.

- GetQueryResultsMonitorTopContributors
- GetQueryResultsMonitorsTopContributors
- GetQueryResultsWorkloadInsightsTopContributors
- GetQueryResultsWorkloadInsightsTopContributorsData
- GetQueryStatusMonitorTopContributors
- GetQueryStatusMonitorsTopContributors
- GetQueryStatusWorkloadInsightsTopContributors

- GetQueryStatusWorkloadInsightsTopContributorsData
- StartQueryMonitorTopContributors
- StartQueryMonitorsTopContributors
- StartQueryWorkloadInsightsTopContributors
- StartQueryWorkloadInsightsTopContributorsData
- StopQueryMonitorTopContributors
- StopQueryMonitorsTopContributors
- StopQueryWorkloadInsightsTopContributors
- StopQueryWorkloadInsightsTopContributorsData

The following example shows a CloudTrail log entry that demonstrates the GetQueryResultsMonitorsTopContributors action.


```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",
         "userName": "SAMPLE_NAME"
      },
      "attributes": {
      "creationDate": "2024-11-03T15:43:27Z",
      "mfaAuthenticated": "false"
      }
    }
},
  "eventTime": "2024-11-15T14:08:04Z",
  "eventSource": "networkflowmonitor.amazonaws.com",
  "eventName": "GetQueryResultsMonitorTopContributors",
  "awsRegion": "us-east-1",
```

```
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "errorCode": "AccessDenied",
  "requestParameters": {
    "QueryId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEQuery,
    "MaxResults": "20",
    "MonitorName": "TestMonitor"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
  "readOnly": true,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::NetworkFlowMonitor::Monitor",
      "ARN": "arn:aws:networkflowmonitor:us-east-1:123456789012:monitor/TestMonitor"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "000000000000",
  "eventCategory": "Data"
}
```


The following example shows a CloudTrail log entry that demonstrates the GetQueryResultsWorkloadInsightsTopContributors action.


```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::000000000000:role/Admin",
        "accountId": "123456789012",
         "userName": "SAMPLE_NAME"
```

```
      },
      "attributes": {
      "creationDate": "2024-11-03T15:43:27Z",
      "mfaAuthenticated": "false"
      }
    }
},
  "eventTime": "2024-11-15T14:08:04Z",
  "eventSource": "networkflowmonitor.amazonaws.com",
  "eventName": "GetQueryResultsWorkloadInsightsTopContributorsData",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
  "errorCode": "AccessDenied",
  "requestParameters": {
    "QueryId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEQuery",
    "ScopeId": "a1b2c3d4-5678-90ab-cdef-EXAMPLEScope"
  },
  "responseElements": null,
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
  "readOnly": true,
  "resources": [
    {
      "accountId": "496383180932",
      "type": "AWS::NetworkFlowMonitor::Scope",
      "ARN": "arn:aws:networkflowmonitor:us-east-1:123456789012:scope/
a1b2c3d4-5678-90ab-cdef-EXAMPLEScope"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "000000000000",
  "eventCategory": "Data"
}
```


Internet Monitor in CloudTrail

Internet Monitor supports logging the following actions as events in CloudTrail log files.

- CreateMonitor
- DeleteMonitor

- GetHealthEvent
- GetMonitor
- GetQueryResults
- GetQueryStatus
- ListHealthEvents
- ListInternetEvents
- ListMonitors
- ListTagsForResource
- StartQuery
- StopQuery
- TagResource
- UntagResource
- UpdateMonitor

Example: Internet Monitor log file entries

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `ListMonitors` action.

```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
```

```
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:18Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "ListMonitors",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```

The following example shows a CloudTrail Internet Monitor log entry that demonstrates the `CreateMonitor` action.

```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::000000000000:assumed-role/role_name",
            "accountId":"123456789012",
            "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::000000000000:role/Admin",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
                },
                "webIdFederationData": {},
                "attributes": {
```

```
                    "creationDate": "2022-10-11T17:25:41Z",
                    "mfaAuthenticated": "false"
                }
            }
        },
        "eventTime": "2022-10-11T17:30:08Z",
        "eventSource": "internetmonitor.amazonaws.com",
        "eventName": "CreateMonitor",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
        "requestParameters": {
            "MonitorName": "TestMonitor",
            "Resources": ["arn:aws:ec2:us-east-2:444455556666:vpc/vpc-febc0b95"],
            "ClientToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": {
            "Arn": "arn:aws:internetmonitor:us-east-2:444455556666:monitor/ct-
onboarding-test",
            "Status": "PENDING"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
    }
```


CloudWatch Synthetics information in CloudTrail

CloudWatch Synthetics supports logging the following actions as events in CloudTrail log files:

- AssociateResource
- CreateCanary
- CreateGroup
- DeleteCanary
- DeleteGroup
- DescribeCanaries

- DescribeCanariesLastRun
- DescribeRuntimeVersions
- DisassociateResource
- GetCanary
- GetCanaryRuns
- GetGroup
- ListAssociatedGroups
- ListGroupResources
- ListGroups
- ListTagsForResource
- StartCanary
- StartCanaryDryRun
- StopCanary
- TagResource
- UntagResource
- UpdateCanary

> **Note**
>
> When you add, update, or remove replica locations for a multilocation canary, related activity appears in CloudTrail in each affected replica Region under the event name `ReplicateCanary`.

Example: CloudWatch Synthetics log file entries

The following example shows a CloudTrail Synthetics log entry that demonstrates the `DescribeCanaries` action.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
```

```
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
                "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "DescribeCanaries",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "201ed5f3-15db-4f87-94a4-123456789",
    "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

The following example shows a CloudTrail Synthetics log entry that demonstrates the `UpdateCanary` action.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
```

```
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
               "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:47Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "UpdateCanary",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Schedule": {
            "Expression": "rate(1 minute)"
        },
        "name": "sample_canary_name",
        "Code": {
            "Handler": "myOwnScript.handler",
            "ZipFile": "SAMPLE_ZIP_FILE"
        }
    },
    "responseElements": null,
    "requestID": "fe4759b0-0849-4e0e-be71-1234567890",
    "eventID": "9dc60c83-c3c8-4fa5-bd02-1234567890",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

The following example shows a CloudTrail Synthetics log entry that demonstrates the `GetCanaryRuns` action.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:assumed-role/role_name",
        "accountId":"123456789012",
        "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111222333444:role/Administrator",
       "accountId":"123456789012",
                "userName": "SAMPLE_NAME"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-04-08T21:43:24Z"
            }
        }
    },
    "eventTime": "2020-04-08T23:06:30Z",
    "eventSource": "synthetics.amazonaws.com",
    "eventName": "GetCanaryRuns",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.590
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.212-b03
 java/1.8.0_212 vendor/Oracle_Corporation",
    "requestParameters": {
        "Filter": "TIME_RANGE",
        "name": "sample_canary_name",
        "FilterValues": [
            "2020-04-08T23:00:00.000Z",
            "2020-04-08T23:10:00.000Z"
        ]
    },
    "responseElements": null,
```

```
    "requestID": "2f56318c-cfbd-4b60-9d93-1234567890",
    "eventID": "52723fd9-4a54-478c-ac55-1234567890",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CloudWatch RUM information in CloudTrail

CloudWatch RUM supports logging the following actions as events in CloudTrail log files:

- BatchCreateRumMetricDefinitions
- BatchDeleteRumMetricDefinitions
- BatchGetRumMetricDefinitions
- CreateAppMonitor
- DeleteAppMonitor
- DeleteResourcePolicy
- DeleteRumMetricsDestination
- GetAppMonitor
- GetAppMonitorData
- GetResourcePolicy
- ListAppMonitors
- ListRumMetricsDestinations
- ListTagsForResource
- PutResourcePolicy
- PutRumMetricsDestination
- TagResource
- UntagResource
- UpdateAppMonitor
- UpdateRumMetricDefinition

Example: CloudWatch RUM log file entries

This section contains example CloudTrail entries for some CloudWatch RUM APIs.

The following example shows a CloudTrail log entry that demonstrates the CreateAppMonitor action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:02:57Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "CreateAppMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "54.240.198.39",
    "userAgent": "aws-internal/3 aws-sdk-java/1.12.641
Linux/5.10.219-186.866.amzn2int.x86_64 OpenJDK_64-Bit_Server_VM/25.402-b08
java/1.8.0_402 vendor/Oracle_Corporation cfg/retry-mode/standard",
    "requestParameters": {
        "CustomEvents": {
            "Status": "ENABLED"
        },
        "CwLogEnabled": true,
        "Domain": "*.github.io",
        "AppMonitorConfiguration": {
            "SessionSampleRate": 1,
            "IncludedPages": [],
            "ExcludedPages": [],
```

```
            "Telemetries": [
                "performance",
                "errors",
                "http"
            ],
            "EnableXRay": false,
            "AllowCookies": true,
            "IdentityPoolId": "us-east-1:c81b9a1c-a5c9-4de5-8585-eb8df04e66f0"
        },
        "Tags": {
            "TestAppMonitor": ""
        },
        "Name": "TestAppMonitor"
    },
    "responseElements": {
        "Id": "65a8cc63-4ae8-4f2c-b5fc-4a54ef43af51"
    },
    "requestID": "cf7c30ad-25d3-4274-bab1-39c95a558007",
    "eventID": "2d43cc69-7f89-4f1a-95ae-0fc7e9b9fb3b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

The following example shows a CloudTrail log entry that demonstrates the PutRumMetricsDestination action.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
```

```
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:22:22Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "PutRumMetricsDestination",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.put-rum-metrics-destination",
    "requestParameters": {
        "Destination": "CloudWatch",
        "AppMonitorName": "TestAppMonitor"
    },
    "responseElements": null,
    "requestID": "9b03fcce-b3a2-44fc-b771-900e1702998a",
    "eventID": "6250f9b7-0505-4f96-9668-feb64f82de5b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```


The following example shows a CloudTrail log entry that demonstrates the BatchCreateRumMetricsDefinitions action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID",
        "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
        "accountId": "777777777777",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                "accountId": "777777777777",
                "userName": "USERNAME_EXAMPLE"
            },
            "attributes": {
                "creationDate": "2024-07-23T16:48:47Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-07-23T18:23:11Z",
    "eventSource": "rum.amazonaws.com",
    "eventName": "BatchCreateRumMetricDefinitions",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.142",
    "userAgent": "aws-cli/2.13.25 Python/3.11.5 Linux/5.10.219-186.866.amzn2int.x86_64
 exe/x86_64.amzn.2 prompt/off command/rum.batch-create-rum-metric-definitions",
    "requestParameters": {
        "Destination": "CloudWatch",
        "MetricDefinitions": [
            {
                "Name": "NavigationToleratedTransaction",
                "Namespace": "AWS/RUM",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"]},
\"event_type\":[\"com.amazon.rum.performance_navigation_event\"],\"event_details\":
 {\"duration\": [{\"numeric\": [\"<=\",2000,\"<\",8000]}]}}"
            },
            {
                "Name": "HttpErrorCount",
                "DimensionKeys": {
                    "metadata.browserName": "BrowserName",
                    "metadata.countryCode": "CountryCode"
                },
                "EventPattern": "{\"metadata\":{\"browserName\":[\"Chrome\"],
 \"countryCode\":[\"US\"]},\"event_type\":[\"com.amazon.rum.http_event\"]}"
            }
        ],
        "AppMonitorName": "TestAppMonitor"
    },
    "responseElements": {
```

```
        "Errors": [],
        "MetricDefinitions": []
    },
    "requestID": "b14c5eda-f107-48e5-afae-1ac20d0962a8",
    "eventID": "001b55c6-1de1-48c0-a236-31096dffe249",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "777777777777",
    "eventCategory": "Management"
}
```

CloudWatch RUM data plane events in CloudTrail

CloudTrail can capture API activities related to the CloudWatch RUM data plane operation PutRumEvents.

Data events, also known as data plane operations, give you insight into the resource operations performed on or within a resource. Data events are often high-volume activities.

To enable logging of the **PutRumEvents** data events in CloudTrail files, you'll need to enable the logging of data plane API activity in CloudTrail. See Logging data events for trails for more information.

Data plane events can be filtered by resource type. Because there are additional costs for using data events in CloudTrail, filtering by resource allows you to have more control over what you log and pay for.

Using the information that CloudTrail collects, you can identify a specific request to the CloudWatch RUM **PutRumEvents** API, the IP address of the requester, the requester's identity, and the date and time of the request. Logging the **PutRumEvents** API using CloudTrail helps you enable operational and risk auditing, governance, and compliance of your AWS account.

The following example shows a CloudTrail log entry that demonstrates the PutRumEvents action.

```
{
 "Records": [
     {
         "eventVersion": "1.09",
         "userIdentity": {
             "type": "AssumedRole",
             "principalId": "EXAMPLE_PRINCIPAL_ID",
```

```
            "arn": "arn:aws:sts::777777777777:assumed-role/EXAMPLE",
            "accountId": "777777777777",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "sessionContext": {
                "sessionIssuer": {
                    "type": "Role",
                    "principalId": "EXAMPLE_PRINCIPAL_ID",
                    "arn": "arn:aws:iam::777777777777:role/EXAMPLE",
                    "accountId": "777777777777",
                    "userName": "USERNAME_EXAMPLE"
                },
                "attributes": {
                    "creationDate": "2024-05-16T20:32:39Z",
                    "mfaAuthenticated": "false"
                }
            },
            "invokedBy": "AWS Internal"
        },
        "eventTime": "2024-05-16T20:32:42Z",
        "eventSource": "rum.amazonaws.com",
        "eventName": "PutRumEvents",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "AWS Internal",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "id": "73ddbd81-1234-5678-b246-123456789",
            "batchId": "123456-3dd0-4ada-b246-123456789",
            "appMonitorDetails": {
                "name": "APP-MONITOR-NAME",
                "id": "123456-3dd0-4ada-b246-123456789",
                "version": "1.0.0"
            },
            "userDetails": {
                "userId": "73ddbd81-1111-9999-b246-123456789",
                "sessionId": "a1b2c3456-15db-4f87-6789-123456789"
            },
            "rumEvents": [
                {
                    "id": "201f367a-15db-1234-94a4-123456789",
                    "timestamp": "May 16, 2024, 8:32:20 PM",
                    "type": "com.amazon.rum.dom_event",
                    "metadata": "{}",
                    "details": "{}"
                }
```

```
            ]
        },
        "responseElements": null,
        "requestID": "201ed5f3-15db-4f87-94a4-123456789",
        "eventID": "73ddbd81-3dd0-4ada-b246-123456789",
        "readOnly": false,
        "resources": [
            {
                "accountId": "777777777777",
                "type": "AWS::RUM::AppMonitor",
                "ARN": "arn:aws:rum:us-east-1:777777777777:appmonitor/
APPMONITOR_NAME_EXAMPLE"
            }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "777777777777",
        "eventCategory": "Data"
    }
 ]
}
```

Network Synthetic Monitor information in CloudTrail

Network Synthetic Monitor supports logging the following actions as events in CloudTrail log files:

- CreateMonitor
- CreateProbe
- DeleteMonitor
- DeleteProbe
- GetMonitor
- GetProbe
- ListMonitors
- ListTagsForResource
- TagResource
- UntagResource
- UpdateMonitor
- UpdateProbe

Example: Network Synthetic Monitor log file entries

The following example shows a Network Synthetic Monitor CloudTrail log entry that demonstrates the `CreateMonitor` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:assumed-role/role_name",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "networksynthetics.amazonaws.com",
    "eventName": "CreateMonitor",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
        "MonitorName": "TestNetworkSyntheticMonitor",
        "ClientToken": "33551db7-1618-4aab-cdef-EXAMPLE33333"
    },
    "responseElements": {
        "MonitorArn": "arn:aws:networksynthetics:us-east-1:111122223333:monitor/
TestNetworkSyntheticMonitor",
        "MonitorName": "TestNetworkSyntheticMonitor",
        "MonitorStatus": "ACTIVE"
    },
```

```
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

CloudWatch Observability Access Manager information in CloudTrail

CloudWatch Observability Access Manager supports logging the following actions as events in CloudTrail log files:

- CreateLink
- CreateSink
- DeleteLink
- DeleteSink
- GetLink
- GetSink
- GetSinkPolicy
- ListAttachedLinks
- ListLinks
- ListSinks
- ListTagsForResource
- PutSinkPolicy
- TagResource
- UntagResource
- UpdateLink

Example: CloudWatch Observability Access Manager log file entries

The following example shows a CloudWatch Observability Access Manager CloudTrail log entry that demonstrates the `CreateSink` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:assumed-role/role_name",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "oam.amazonaws.com",
    "eventName": "CreateSink",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
        "Name": "TestObservabilitySink"
    },
    "responseElements": {
        "Arn": "arn:aws:oam:us-east-1:111122223333:sink/TestObservabilitySink",
        "Id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "Name": "TestObservabilitySink"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
```

```
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

CloudWatch Observability Admin information in CloudTrail

CloudWatch Observability Admin supports logging the following actions as events in CloudTrail log files:

- GetTelemetryEvaluationStatus
- GetTelemetryEvaluationStatusForOrganization
- ListResourceTelemetry
- ListResourceTelemetryForOrganization
- StartTelemetryEvaluation
- StartTelemetryEvaluationForOrganization
- StopTelemetryEvaluation
- StopTelemetryEvaluationForOrganization

Example: CloudWatch Observability Admin log file entries

The following example shows a CloudWatch Observability Admin CloudTrail log entry that demonstrates the `StartTelemetryEvaluation` action.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:assumed-role/role_name",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "observabilityadmin.amazonaws.com",
    "eventName": "StartTelemetryEvaluation",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {},
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


CloudWatch Application Signals information in CloudTrail

CloudWatch Application Signals supports logging the following actions as events in CloudTrail log files:

- BatchGetServiceLevelObjectiveBudgetReport
- BatchUpdateExclusionWindows
- CreateServiceLevelObjective
- DeleteServiceLevelObjective
- GetService
- GetServiceLevelObjective
- ListServiceDependencies

- ListServiceDependents
- ListServiceLevelObjectives
- ListServiceOperations
- ListServices
- ListTagsForResource
- StartDiscovery
- TagResource
- UntagResource
- UpdateServiceLevelObjective

Example: CloudWatch Application Signals log file entries

The following example shows a CloudWatch Application Signals CloudTrail log entry that demonstrates the `CreateServiceLevelObjective` action.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:assumed-role/role_name",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "applicationsignals.amazonaws.com",
```

```
    "eventName": "CreateServiceLevelObjective",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
        "Name": "TestSLO",
        "Description": "Test Service Level Objective"
    },
    "responseElements": {
        "Arn": "arn:aws:applicationsignals:us-east-1:111122223333:slo/TestSLO"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


CloudWatch Application Insights information in CloudTrail

CloudWatch Application Insights supports logging the following actions as events in CloudTrail log files:

- AddWorkload
- CreateApplication
- CreateComponent
- CreateLogPattern
- DeleteApplication
- DeleteComponent
- DeleteLogPattern
- DescribeApplication
- DescribeComponent
- DescribeComponentConfiguration
- DescribeComponentConfigurationRecommendation
- DescribeLogPattern

- DescribeObservation
- DescribeProblem
- DescribeProblemObservations
- DescribeWorkload
- ListApplications
- ListComponents
- ListConfigurationHistory
- ListLogPatterns
- ListLogPatternSets
- ListProblems
- ListTagsForResource
- ListWorkloads
- RemoveWorkload
- TagResource
- UntagResource
- UpdateApplication
- UpdateComponent
- UpdateComponentConfiguration
- UpdateLogPattern
- UpdateProblem
- UpdateWorkload

Example: CloudWatch Application Insights log file entries

The following example shows a CloudWatch Application Insights CloudTrail log entry that demonstrates the `CreateApplication` action.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::111122223333:assumed-role/role_name",
```

```
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "SAMPLE_NAME"
            },
            "attributes": {
                "creationDate": "2024-11-03T15:43:27Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-11-03T15:58:11Z",
    "eventSource": "applicationinsights.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "requestParameters": {
        "ResourceGroupName": "TestApplicationResourceGroup"
    },
    "responseElements": {
        "ApplicationInfo": {
            "ResourceGroupName": "TestApplicationResourceGroup",
            "LifeCycle": "ACTIVE"
        }
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Tagging your Amazon CloudWatch resources

A *tag* is a custom attribute label that you or AWS assigns to an AWS resource.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging. You can assign the same tag to resources from different services to indicate the resources are related. For example, you can assign the tag that you assigned to a CloudWatch rule to an EC2 instance.

Tags have two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

The following sections provide more information about tags for CloudWatch.

Supported resources in CloudWatch

The following resources in CloudWatch support tagging:

- Alarms – You can tag alarms using the tag-resource AWS CLI command and the TagResource API. You can also view and manage your alarm tags using the *Alarms* details page in the CloudWatch console.
- Canaries – You can tag canaries using the CloudWatch console. For more information, see Creating a canary.
- Contributor Insights rules – You can tag Contributor Insights rules when you create them by using the put-insight-rule AWS CLI command and the PutInsightRule API. You can add tags to existing rules by using the tag-resource AWS CLI command and the TagResource API.
- Dashboards – You can tag dashboards when you create them by using the put-dashboard AWS CLI command and the PutDashboard API. You can add tags to existing dashboards by using the tag-resource AWS CLI command and the TagResource API.
- Metric streams – You can tag metric streams when you create them by using the put-metric-stream AWS CLI command and the PutMetricStream API. You can add tags to existing metric streams by using the tag-resource AWS CLI command and the TagResource API.

For information about adding and managing tags, see Managing tags.

Managing tags

Tags consist of the `Key` and `Value` properties on a resource. You can use the CloudWatch console, the AWS CLI, or the CloudWatch API to add, edit, or delete the values for these properties. For information about working with tags, see the following:

- TagResource, UntagResource, and ListTagsForResource in the *Amazon CloudWatch API Reference*
- tag-resource, untag-resource, and list-tags-for-resource in the *Amazon CloudWatch CLI Reference*
- Working with Tag Editor in the *Resource Groups User Guide*

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with CloudWatch resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: ***. : + = @ _ / -*** (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter` and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is prohibited for tags because it's reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix don't count against your tags per resource limit.

CloudWatch service quotas

Amazon CloudWatch provides monitoring and observability for your AWS resources and applications. To help ensure optimal performance and prevent abuse, CloudWatch imposes service quotas on various aspects of its functionality. This chapter outlines the key quotas for CloudWatch services, including metrics, alarms, API requests, and notifications. Understanding these quotas is crucial for effectively planning and managing your CloudWatch usage.

> **Note**
>
> For some AWS services including CloudWatch, you can use the CloudWatch usage metrics to visualize your current service usage on CloudWatch graphs and dashboards. You can use a CloudWatch metric math function to display the service quotas for those resources on your graphs. You can also configure alarms that alert you when your usage approaches a service quota. For more information, see Visualizing your service quotas and setting alarms.

Topics

- CloudWatch
- CloudWatch investigations (Amazon AI Operations)
- CloudWatch Application Insights
- CloudWatch Application Signals
- Internet Monitor
- Network Flow Monitor
- Network Synthetic Monitor
- CloudWatch Observability Access Manager
- CloudWatch Observability Admin
- CloudWatch Omni
- CloudWatch RUM
- Managing your CloudWatch service quotas

CloudWatch

This section details the service quotas specifically for core CloudWatch functionality. These quotas cover aspects such as metrics, alarms, API requests, and dashboards. Familiarizing yourself with these limits helps you optimize your CloudWatch configuration and avoid potential throttling or service disruptions.

Name	Default	Adjustable	Description
Canary limit	Each supported Region: 500	Yes	The maximum number of canaries per account per region.
Number of Alarm Mute Rules	Each supported Region: 2,000	No	The maximum number of Alarm Mute Rules that you can have in this account in the current region.
Number of Contributor Insights rules	Each supported Region: 100	Yes	The maximum number of Contributor Insights rules you can have in this account.
Number of Metrics Insights alarms	Each supported Region: 200	No	The maximum number of Metrics Insights alarms that you can have in this account in the current region.
Number of PromQL alarms	Each supported Region: 15,000	No	The maximum number of PromQL alarms that you can have in this account in the current region.
Rate of ANOMALY_DETECTION_BAND usage in GetMetricData	Each supported Region: 1,000	No	The maximum number of times the ANOMALY_D

Name	Default	Adjustable	Description
			ETECTION_BAND function can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of DB_PERF_INSIGHTS usage in GetMetricData	Each supported Region: 4	No	The maximum number of times the DB_PERF_INSIGHTS function can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of DeleteAlarmMuteRule requests	Each supported Region: 3 per second	No	The maximum number of DeleteAlarmMuteRule requests that you can make, per second, in this account in the current region.
Rate of DeleteAlarms requests	Each supported Region: 3 per second	No	The maximum number of DeleteAlarms requests that you can make, per second, in this account in the current region.
Rate of DeleteAnomalyDetector requests	Each supported Region: 5 per second	No	The maximum number of DeleteAnomalyDetector requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of DeleteDashboards requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteDashboards requests that you can make, per second, in this account in the current region.
Rate of DeleteInsightRules requests	Each supported Region: 5 per second	No	The maximum number of DeleteInsightRules requests you can make per second in this account.
Rate of DeleteMetricStream requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteMetricStream requests that you can make, per second, in this account in the current region.
Rate of DescribeAlarmContributors requests	Each supported Region: 9 per second	Yes	The maximum number of DescribeAlarmContributors requests that you can make, per second, in this account in the current region.
Rate of DescribeAlarmHistory requests	Each supported Region: 20 per second	No	The maximum number of DescribeAlarmHistory requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of DescribeAlarms requests	Each supported Region: 9 per second	Yes	The maximum number of DescribeAlarms requests that you can make, per second, in this account in the current region.
Rate of DescribeAlarmsForMetric requests	Each supported Region: 9 per second	No	The maximum number of DescribeAlarmsForMetric requests that you can make, per second, in this account in the current region.
Rate of DescribeAnomalyDetectors requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeAnomalyDetectors requests that you can make, per second, in this account in the current region.
Rate of DescribeInsightRules requests	Each supported Region: 20 per second	No	The maximum number of DescribeInsightRules requests you can make per second in this account.
Rate of DisableAlarmActions requests	Each supported Region: 3 per second	No	The maximum number of DisableAlarmActions requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of DisableInsightRules requests	Each supported Region: 1 per second	No	The maximum number of DisableInsightRules requests you can make per second in this account.
Rate of EnableAlarmActions requests	Each supported Region: 3 per second	No	The maximum number of EnableAlarmActions requests that you can make, per second, in this account in the current region.
Rate of EnableInsightRules requests	Each supported Region: 1 per second	No	The maximum number of EnableInsightRules requests you can make per second in this account.
Rate of GetAlarmMuteRule requests	Each supported Region: 20 per second	Yes	The maximum number of GetAlarmMuteRule requests that you can make, per second, in this account in the current region.
Rate of GetDashboard requests	Each supported Region: 10 per second	Yes	The maximum number of GetDashboard requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of GetInsightRuleReport requests	Each supported Region: 20 per second	Yes	The maximum number of GetInsightRuleReport requests you can make, per second in this account.
Rate of GetMetricData datapoints for metrics older than three hours	Each supported Region: 396,000	No	The maximum number of GetMetricData datapoints that you can fetch, per second, for a request with a StartTime of more than three hours in this account in the current region.
Rate of GetMetricData datapoints for the last three hours of metrics	Each supported Region: 180,000	No	The maximum number of GetMetricData datapoints that you can fetch, per second, for a request with a StartTime of less than or equal to three hours in this account in the current region.
Rate of GetMetricData datapoints using Metrics Insights	Each supported Region: 4,300,000	No	The maximum number of GetMetricData datapoints that you can fetch using Metrics Insights, per second, for a request with a StartTime of less than or equal to three hours in this account in the current region.

Name	Default	Adjustable	Description
Rate of GetMetricData requests	Each supported Region: 500 per second	Yes	The maximum number of GetMetricData requests that you can make, per second, in this account in the current region.
Rate of GetMetricStatistics requests	Each supported Region: 400 per second	Yes	The maximum number of GetMetricStatistics requests that you can make, per second, in this account in the current region.
Rate of GetMetricStream requests	Each supported Region: 10 per second	Yes	The maximum number of GetMetricStream requests that you can make, per second, in this account in the current region.
Rate of GetMetricWidgetImage requests	Each supported Region: 20 per second	Yes	The maximum number of GetMetricWidgetImage requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of INSIGHT_RULE_METRIC usage in GetMetricData	Each supported Region: 20	No	The maximum number of times the INSIGHT_RULE_METRIC function can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of LAMBDA usage in GetMetricData	Each supported Region: 5	No	The maximum number of times the LAMBDA function can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of ListAlarmMuteRules requests	Each supported Region: 50 per second	Yes	The maximum number of ListAlarmMuteRules requests that you can make, per second, in this account in the current region.
Rate of ListDashboards requests	Each supported Region: 10 per second	Yes	The maximum number of ListDashboards requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of ListMetricStreams requests	Each supported Region: 10 per second	Yes	The maximum number of ListMetricStreams requests that you can make, per second, in this account in the current region.
Rate of ListMetrics requests	Each supported Region: 25 per second	Yes	The maximum number of ListMetrics requests that you can make, per second, in this account in the current region.
Rate of ListTagsForResource requests	Each supported Region: 10 per second	No	The maximum number of ListTagsForResource requests that you can make, per second, in this account in the current region.
Rate of Metrics Insights usage in GetMetricData	Each supported Region: 10	No	The maximum number of times Metrics Insights can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of PutAlarmMuteRule requests	Each supported Region: 3 per second	Yes	The maximum number of PutAlarmMuteRule requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of PutAnomalyDetector requests	Each supported Region: 10 per second	Yes	The maximum number of PutAnomalyDetector requests that you can make, per second, in this account in the current region.
Rate of PutCompositeAlarm requests	Each supported Region: 3 per second	Yes	The maximum number of PutCompositeAlarm requests that you can make, per second, in this account in the current region.
Rate of PutDashboard requests	Each supported Region: 10 per second	Yes	The maximum number of PutDashboard requests that you can make, per second, in this account in the current region.
Rate of PutInsightRule requests	Each supported Region: 5 per second	No	The maximum number of PutInsightRule requests you can make, per second in this account.
Rate of PutLogAlarm requests	Each supported Region: 3 per second	No	The maximum number of PutLogAlarm requests that you can make, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of PutMetricAlarm requests	Each supported Region: 3 per second	Yes	The maximum number of PutMetricAlarm requests that you can make, per second, in this account in the current region.
Rate of PutMetricData requests	Each supported Region: 500 per second	Yes	The maximum number of PutMetricData requests that you can make, per second, in this account in the current region.
Rate of PutMetricStream requests	Each supported Region: 10 per second	Yes	The maximum number of PutMetricStream requests that you can make, per second, in this account in the current region.
Rate of SEARCH usage in GetMetricData	Each supported Region: 50	No	The maximum number of times the SEARCH function can be used in all GetMetricData requests, per second, in this account in the current region.
Rate of SERVICE_QUOTA usage in GetMetricData	Each supported Region: 1,000	No	The maximum number of times the SERVICE_QUOTA function can be used in all GetMetricData requests, per second, in this account in the current region.

Name	Default	Adjustable	Description
Rate of SetAlarmState requests	Each supported Region: 3 per second	No	The maximum number of SetAlarmState requests that you can make, per second, in this account in the current region.
Rate of StartMetricStreams requests	Each supported Region: 10 per second	Yes	The maximum number of StartMetricStreams requests that you can make, per second, in this account in the current region.
Rate of StopMetricStreams requests	Each supported Region: 10 per second	Yes	The maximum number of StopMetricStreams requests that you can make, per second, in this account in the current region.
Rate of TagResource requests	Each supported Region: 20 per second	No	The maximum number of TagResource requests that you can make, per second, in this account in the current region.
Rate of UntagResource requests	Each supported Region: 20 per second	No	The maximum number of UntagResource requests that you can make, per second, in this account in the current region.

CloudWatch investigations (Amazon AI Operations)

CloudWatch investigations (Amazon AI Operations) enables intelligent problem detection and root cause analysis. This section outlines the quotas related to concurrent investigations, investigation groups, and AI-assisted analyses.

Name	Default	Adjuse	Description
Concurrent active investigations	Each supported Region: 2	No	The maximum number of concurrent investigations with active AI analysis per account per region.
Investigation groups	Each supported Region: 1	No	The maximum number of investigation groups that can be created per account in a region.
Monthly investigations	Each supported Region: 150	No	The maximum number of AI-assisted investigations per month in this account in the current region.

CloudWatch Application Insights

CloudWatch Application Insights helps you monitor your applications and troubleshoot issues quickly. This section presents the quotas for CloudWatch Application Insights, including limits on API requests, applications, log streams, and metrics. Knowing these quotas allows you to effectively plan your application monitoring setup.

Resource	Default quota	Adjustable
API requests	All API actions are throttled to 5 TPS	No

Resource	Default quota	Adjustable
Resource Group applications	100 per account	No
Account applications	1 per account	No
Log Streams	5 per resource	No
Observations per problem	20 per dashboard 40 per DescribeProblemObservations action	No
Metrics	60 per resource	No

CloudWatch Application Signals

CloudWatch Application Signals provides deep insights into your application's performance and dependencies. This section covers the quotas for Application Signals, including API request limits and service-level objective (SLO) constraints. Understanding these quotas is crucial for implementing effective application performance monitoring.

Name	Default	Adjuse	Description
Number of SLOs per Region	Each supported Region: 250	Yes	The maximum number of SLOs that you can have in this account in the current region.
Number of SLOs per Service	Each supported Region: 100	Yes	The maximum number of SLOs that you can have per service in the current region.
Rate of BatchGetServiceLevelObjectiveBudgetReport requests	Each supported Region: 10 per second	Yes	The maximum number of BatchGetServiceLevelObjectiveBudgetR

Name	Default	Adjustable	Description
			eport requests you can make per second in this Region.
Rate of BatchUpdateExclusionWindows requests	Each supported Region: 5 per second	Yes	The maximum number of BatchUpdateExclusionWindows requests you can make per second in this Region.
Rate of CreateServiceLevelObjective requests	Each supported Region: 5 per second	Yes	The maximum number of CreateServiceLevelObjective requests you can make per second in this Region.
Rate of DeleteGroupingConfiguration requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteGroupingConfiguration requests you can make per second in this Region.
Rate of DeleteServiceLevelObjective requests	Each supported Region: 5 per second	Yes	The maximum number of DeleteServiceLevelObjective requests you can make per second in this Region.
Rate of GetService requests	Each supported Region: 10 per second	Yes	The maximum number of GetService requests you can make per second in this Region.

Name	Default	Adjustable	Description
Rate of GetServiceLevelObjective requests	Each supported Region: 5 per second	Yes	The maximum number of GetServiceLevelObjective requests you can make per second in this Region.
Rate of ListAuditFindings requests	Each supported Region: 1 per second	Yes	The maximum number of ListAuditFindings requests you can make per second in this Region.
Rate of ListEntityEvents requests	Each supported Region: 10 per second	Yes	The maximum number of ListEntityEvents requests you can make per second in this Region.
Rate of ListGroupingAttributeDefinitions requests	Each supported Region: 10 per second	Yes	The maximum number of ListGroupingAttributeDefinitions requests you can make per second in this Region.
Rate of ListServiceDependencies requests	Each supported Region: 10 per second	Yes	The maximum number of ListServiceDependencies requests you can make per second in this Region.
Rate of ListServiceDependents requests	Each supported Region: 10 per second	Yes	The maximum number of ListServiceDependents requests you can make per second in this Region.

Name	Default	Adjustable	Description
Rate of ListServiceLevelObjectiveExclusionWindows requests	Each supported Region: 5 per second	Yes	The maximum number of ListServiceLevelObjectiveExclusionWindows requests you can make per second in this Region.
Rate of ListServiceLevelObjectives requests	Each supported Region: 5 per second	Yes	The maximum number of ListServiceLevelObjectives requests you can make per second in this Region.
Rate of ListServiceOperations requests	Each supported Region: 10 per second	Yes	The maximum number of ListServiceOperations requests you can make per second in this Region.
Rate of ListServiceStates requests	Each supported Region: 5 per second	Yes	The maximum number of ListServiceStates requests you can make per second in this Region.
Rate of ListServices requests	Each supported Region: 10 per second	Yes	The maximum number of ListServices requests you can make per second in this Region.
Rate of ListTagsForResource requests	Each supported Region: 5 per second	Yes	The maximum number of ListTagsForResource requests you can make per second in this Region.

Name	Default	Adjustable	Description
Rate of PutGroupingConfiguration requests	Each supported Region: 10 per second	Yes	The maximum number of PutGroupingConfiguration requests you can make per second in this Region.
Rate of StartDiscovery requests	Each supported Region: 5 per second	Yes	The maximum number of StartDiscovery requests you can make per second in this Region.
Rate of TagResource requests	Each supported Region: 5 per second	Yes	The maximum number of TagResource requests you can make per second in this Region.
Rate of UntagResource requests	Each supported Region: 5 per second	Yes	The maximum number of UntagResource requests you can make per second in this Region.
Rate of UpdateServiceLevelObjective requests	Each supported Region: 5 per second	Yes	The maximum number of UpdateServiceLevelObjective requests you can make per second in this Region.

Internet Monitor

Internet Monitor, a feature of CloudWatch Network Monitoring, helps you understand how internet issues impact your application's performance. This section provides the quotas specific to Internet Monitor, to help you to plan an effective internet performance monitoring strategy.

Name	Default	Adjuse	Description
Days that health events are retained	Each supported Region: 400	No	The number of days that AWS keeps information about a resolved Internet Monitor health event.
Monitors per account per AWS Region	Each supported Region: 50	No	The maximum number of monitors in an account in one AWS Region.
Resources per monitor	Each supported Region: 50	No	The maximum number of resources that a monitor can have.

Network Flow Monitor

Network Flow Monitor, a feature of CloudWatch Network Monitoring, provides near real-time visibility into network performance, such as packet loss and latency, for traffic between Amazon EC2 instances, as well as traffic toward certain other AWS services. It also helps you to determine if AWS issues are causing network performance degradation for your application. This section provides the quotas for Network Flow Monitor, including limits on monitored flows, data retention, and API requests. Understanding these quotas can help you to plan an effective network flow monitoring strategy.

Resource	Default quota	Adjustable?
Scopes per account per AWS Region	1	No
Monitors per account per AWS Region	20	Yes
Local resources per monitor	25	No

Resource	Default quota	Adjustable?
Remote resources per monitor	25	No

Network Synthetic Monitor

Network Synthetic Monitor, a feature of CloudWatch Network Monitoring, enables you to proactively monitor your network endpoints and API operations using configurable tests. This section details the quotas for Network Synthetic Monitor, including limits on synthetic monitors, test frequency, and script execution. These quotas are important for planning your network testing strategy and maintaining reliable endpoint monitoring.

Name	Default	Adjustable	Description
Number of monitors per account per AWS region	Each supported Region: 100	Yes	The maximum number of monitors in an account in one AWS Region.
Number of probes per monitor	Each supported Region: 24	Yes	The maximum number of probes that a monitor can have.
Number of probes per subnet for each monitor	Each supported Region: 4	Yes	The maximum number of probes that a subnet in a monitor can have.

CloudWatch Observability Access Manager

CloudWatch Observability Access Manager allows you to securely share CloudWatch resources across accounts. This section details the quotas for Observability Access Manager, including limits on API requests, source account links, and sinks. Understanding these quotas is essential for implementing cross-account observability effectively.

Name	Default	Adjustable	Description
Links per sink	Each supported Region: 100,000	No	Maximum number of links that can be attached to a sink
Number of links	Each supported Region: 5	No	Maximum number of links in your account
Number of sinks	Each supported Region: 1	No	Maximum number of sinks in your account
Rate of CreateLink requests	Each supported Region: 10 per second	No	Maximum number of CreateLink requests you can make per second, in this account in the current region
Rate of CreateSink requests	Each supported Region: 10 per second	No	Maximum number of CreateSink requests you can make per second, in this account in the current region
Rate of DeleteLink requests	Each supported Region: 10 per second	No	Maximum number of DeleteLink requests you can make per second, in this account in the current region
Rate of DeleteSink requests	Each supported Region: 10 per second	No	Maximum number of DeleteSink requests you can make per second, in this account in the current region

Name	Default	Adjustable	Description
Rate of GetLink requests	Each supported Region: 10 per second	No	Maximum number of GetLink requests you can make per second, in this account in the current region
Rate of GetSink requests	Each supported Region: 10 per second	No	Maximum number of GetSink requests you can make per second, in this account in the current region
Rate of GetSinkPolicy requests	Each supported Region: 10 per second	No	Maximum number of GetSinkPolicy requests you can make per second, in this account in the current region
Rate of ListAttachedLinks requests	Each supported Region: 10 per second	No	Maximum number of ListAttachedLinks requests you can make per second, in this account in the current region
Rate of ListLinks requests	Each supported Region: 10 per second	No	Maximum number of ListLinks requests you can make per second, in this account in the current region

Name	Default	Adjustable	Description
Rate of ListSinks requests	Each supported Region: 10 per second	No	Maximum number of ListSinks requests you can make per second, in this account in the current region
Rate of ListTagsForResource requests	Each supported Region: 10 per second	No	Maximum number of ListTagsForResource requests you can make per second, in this account in the current region
Rate of PutSinkPolicy requests	Each supported Region: 1 per second	No	Maximum number of PutSinkPolicy requests you can make per second, in this account in the current region
Rate of TagResource requests	Each supported Region: 10 per second	No	Maximum number of TagResource requests you can make per second, in this account in the current region
Rate of UntagResource requests	Each supported Region: 10 per second	No	Maximum number of UntagResource requests you can make per second, in this account in the current region

Name	Default	Adjustable	Description
Rate of UpdateLink requests	Each supported Region: 10 per second	No	Maximum number of UpdateLink requests you can make per second, in this account in the current region

CloudWatch Observability Admin

CloudWatch Observability Admin helps you discover, audit, and manage CloudWatch telemetry configurations across an AWS account or organization. It provides a consolidated view of monitoring settings for resources, enabling you to ensure consistent and proper data collection by simplifying the management of telemetry across multiple accounts. Most quotas are adjustable to accommodate enterprise-scale deployments that need higher limits for managing observability rules across large organizations.

Name	Default	Adjustable	Description
CreateCentralizationRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of create-centralization-rule-for-organization calls per second per account/per region
CreateTelemetryPipeline throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of CreateTelemetryPipeline calls per second per account and per Region
CreateTelemetryRule throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of create-telemetry-rule calls per second per account/per region

Name	Default	Adjustable	Description
CreateTelemetryRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of create-telemetry-rule-for-organization calls per second per account/per region
DeleteCentralizationRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of delete-centralization-rule-for-organization calls per second per account/per region
DeleteTelemetryPipeline throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of DeleteTelemetryPipeline calls per second per account and per Region
DeleteTelemetryRule throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of delete-telemetry-rule calls per second per account/per region
DeleteTelemetryRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of delete-telemetry-rule-for-organization calls per second per account/per region
GetCentralizationRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of get-centralization-rule-for-organization calls per second per account/per region

Name	Default	Adjustable	Description
GetTelemetryEnrichmentStatus throttle limit in transactions per second	Each supported Region: 5 per second	Yes	The maximum number of GetTelemetryEnrichmentStatus calls per second per account and per region
GetTelemetryPipeline throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of GetTelemetryPipeline calls per second per account and per Region
GetTelemetryRule throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of get-telemetry-rule calls per second per account/per region
GetTelemetryRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of get-telemetry-rule-for-organization calls per second per account/per region
GetThirdPartyTelemetryEnrichment throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of GetThirdPartyTelemetryEnrichment calls per second per account and per region
ListCentralizationRulesForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of list-centralization-rules-for-organization calls per second per account/per region

Name	Default	Adjustable	Description
ListTagsForResource throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of list-tags-for-resource calls per second per account/per region
ListTelemetryPipelines throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of ListTelemetryPipelines calls per second per account and per Region
ListTelemetryRules throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of list-telemetry-rules calls per second per account/per region
ListTelemetryRulesForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of list-telemetry-rules-for-organization calls per second per account/per region
ListThirdPartyTelemetryEnrichments throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of ListThirdPartyTelemetryEnrichments calls per second per account and per region
Logs Centralization throughput per destination region	Each supported Region: 2.5 Gigabytes per second	No	The maximum organization-level throughput for centralization of logs per destination region

Name	Default	Adjustable	Description
Organization Centralization Rules	Each supported Region: 50	Yes	The maximum number of organization centralization rules that you can create for an organization
Organization Telemetry Rules	Each supported Region: 50	Yes	The maximum number of organization telemetry rules that you can create for an organization
Pipelines for managed OTel metrics experience	Each supported Region: 300	Yes	The maximum number of pipelines for managed OTel metrics experience allowed per account and per Region
Pipelines with a CloudWatch Logs source	Each supported Region: 300	Yes	The maximum number of pipelines with a CloudWatch Logs source allowed per account per Region
Pipelines with a third-party integration source	Each supported Region: 30	Yes	The maximum number of pipelines with a third-party integration source allowed per account per Region
StartTelemetryEnrichment throttle limit in transactions per second	Each supported Region: 1 per second	Yes	The maximum number of StartTelemetryEnrichment calls per second per account and per region

Name	Default	Adjustable	Description
StartThirdPartyTelemetryEnrichment throttle limit in transactions per second	Each supported Region: 1 per second	No	The maximum number of StartThirdPartyTelemetryEnrichment calls per second per account and per region
StopTelemetryEnrichment throttle limit in transactions per second	Each supported Region: 1 per second	Yes	The maximum number of StopTelemetryEnrichment calls per second per account and per region
StopThirdPartyTelemetryEnrichment throttle limit in transactions per second	Each supported Region: 1 per second	No	The maximum number of StopThirdPartyTelemetryEnrichment calls per second per account and per region
TagResource throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of tag-resource calls per second per account/per region
Telemetry Rules	Each supported Region: 20	Yes	The maximum number of telemetry rules that you can create for an account
TestTelemetryPipeline throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of TestTelemetryPipeline calls per second per account and per Region

Name	Default	Adjustable	Description
Third-party telemetry enrichments	Each supported Region: 5	No	The maximum number of third-party telemetry enrichments allowed per account per Region
UntagResource throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of untag-resource calls per second per account/per region
UpdateCentralizationRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of update-centralization-rule-for-organization calls per second per account/per region
UpdateTelemetryPipeline throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of UpdateTelemetryPipeline calls per second per account and per Region
UpdateTelemetryRule throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of update-telemetry-rule calls per second per account/per region
UpdateTelemetryRuleForOrganization throttle limit in transactions per second	Each supported Region: 5	No	The maximum number of update-telemetry-rule-for-organization calls per second per account/per region

Name	Default	Adjustable	Description
ValidateTelemetryPipelineConfiguration throttle limit in transactions per second	Each supported Region: 5 per second	No	The maximum number of ValidateTelemetryPipelineConfiguration calls per second per account and per Region

CloudWatch Omni

CloudWatch Omni is a capability of Amazon CloudWatch that unifies monitoring of your applications, infrastructure, and AI agents in a single observability experience. Omni is built on open standards, uses AI to discover your services and answer questions about your telemetry, and is delivered outside the AWS Management Console. This section details the quotas for CloudWatch Omni, including limits on spaces, grants, access profiles, dashboards, alerts, concurrent queries, and API request rates.

Name	Default	Adjustable	Description
Access profiles per space	Each supported Region: 100	No	The maximum number of access profiles per space
Alerts per space	Each supported Region: 500	No	The maximum number of alerts per space
Concurrent SQL queries per account per region	Each supported Region: 400	No	The maximum number of concurrent SQL queries per account per region
Concurrent query sessions per account per region	Each supported Region: 250	No	The maximum number of concurrent query sessions per account per region

Name	Default	Adjustable	Description
Grants per space	Each supported Region: 500	No	The maximum number of grants per space
Monthly Omni agent credits	Each supported Region: 100	No	The maximum number of Omni agent credits per account per region per month.
Omni dashboards per space	Each supported Region: 200	No	The maximum number of Omni dashboards per space
Omni threads per principal per space	Each supported Region: 200	No	The maximum number of Omni threads per principal per space
Org-wide domain grants per domain	Each supported Region: 100	No	The maximum number of org-wide (domain) grants per domain
Rate of CreateAccessGrant requests	Each supported Region: 5	No	Creates a new rich AccessGrant with optional field filters.
Rate of CreateAccessProfile requests	Each supported Region: 5	No	Creates a new Access Profile within a space.
Rate of CreateAlert requests	Each supported Region: 5	No	Creates a new alert within a space.
Rate of CreateDomain requests	Each supported Region: 5	No	Creates a new domain with identity provider configuration.

Name	Default	Adjustable	Description
Rate of CreateDomainAccessGrantForOrganization requests	Each supported Region: 5	No	Creates an org-scoped ADMIN grant for a principal on the organization domain.
Rate of CreateIntegration requests	Each supported Region: 10	No	Creates an integration with a third-party provider. Returns the integration
Rate of CreateOmniDashboard requests	Each supported Region: 5	No	Creates a new dashboard within a space.
Rate of CreateOneTimeDeepLinkCode requests	Each supported Region: 50	No	Generates a one-time code for deep link authentication.
Rate of CreateSpace requests	Each supported Region: 5	No	Creates a new Space within a domain.
Rate of CreateView requests	Each supported Region: 10	No	Creates a new SQL view.
Rate of DeleteAccessGrant requests	Each supported Region: 5	No	Removes an existing AccessGrant and its associated Cedar policy.
Rate of DeleteAccessProfile requests	Each supported Region: 5	No	Removes an Access Profile.
Rate of DeleteAlert requests	Each supported Region: 5	No	Deletes an alert by its identifier.
Rate of DeleteDomain requests	Each supported Region: 5	No	Removes a domain and all associated resources.

Name	Default	Adjustable	Description
Rate of DeleteDomainAccessGrantForOrganization requests	Each supported Region: 5	No	Deletes an org-scoped grant. Soft-deletes the grant record.
Rate of DeleteDomainForOrganization requests	Each supported Region: 5	No	Deletes an organization-scoped domain. All Spaces must be deleted first.
Rate of DeleteIntegration requests	Each supported Region: 10	No	Deletes an integration. Returns the resulting status.
Rate of DeleteOmniDashboard requests	Each supported Region: 5	No	Removes a dashboard from a space.
Rate of DeleteSpace requests	Each supported Region: 5	No	Removes a Space and all associated resources.
Rate of DeleteView requests	Each supported Region: 10	No	Deletes the specified view.
Rate of GetAccessGrant requests	Each supported Region: 5	No	Retrieves the full detail of a single AccessGrant by ID.
Rate of GetAccessProfile requests	Each supported Region: 5	No	Retrieves an Access Profile with its permissions and access grants.
Rate of GetAlert requests	Each supported Region: 25	No	Retrieves a single alert by its identifier.

Name	Default	Adjustable	Description
Rate of GetContextGraph requests	Each supported Region: 10	No	Queries the context graph with filtering, traversal, and pagination support.
Rate of GetDomain requests	Each supported Region: 5	No	Retrieves a domain by its identifier.
Rate of GetDomainAccessGrantForOrganization requests	Each supported Region: 5	No	Retrieves an org-scoped grant on the organization domain.
Rate of GetDomainForOrganization requests	Each supported Region: 5	No	Retrieves details of an organization-scoped domain.
Rate of GetIntegration requests	Each supported Region: 10	No	Returns the details of a single integration, identified by its identifier.
Rate of GetIntelligenceConfiguration requests	Each supported Region: 10	No	Retrieves the intelligence configuration for the calling account.
Rate of GetOmniDashboard requests	Each supported Region: 5	No	Retrieves a dashboard by ID within a space.
Rate of GetSpace requests	Each supported Region: 5	No	Retrieves details of a specific Space.
Rate of GetSpaceCredentialsForOrganization requests	Each supported Region: 5	No	Vends space-scoped credentials for MA/DA in a member account. IAM_DIRECT only.

Name	Default	Adjustable	Description
Rate of GetTelemetryQueryResults requests	Each supported Region: 20	No	Returns the results for the specified query.
Rate of GetView requests	Each supported Region: 10	No	Returns the definition and metadata of the specified view.
Rate of ListAccessGrants requests	Each supported Region: 5	No	Returns AccessGrants within a space with optional filtering.
Rate of ListAccessProfiles requests	Each supported Region: 5	No	Returns all access profiles in a space.
Rate of ListAlerts requests	Each supported Region: 10	No	Lists alerts within a space, optionally filtered by exact name(s).
Rate of ListDomainAccessGrantsForOrganization requests	Each supported Region: 5	No	Lists org-scoped grants for the organization domain.
Rate of ListDomains requests	Each supported Region: 5	No	Returns all Domains visible to the caller.
Rate of ListIntegrations requests	Each supported Region: 10	No	Lists the integrations in the account, optionally filtered by type, status.
Rate of ListOmniDashboards requests	Each supported Region: 5	No	Returns dashboards within a space.
Rate of ListSpaces requests	Each supported Region: 5	No	Returns all Spaces in the account.

Name	Default	Adjustable	Description
Rate of ListSpacesForOrganization requests	Each supported Region: 5	No	Lists all Spaces across all member accounts in the organization domain.
Rate of ListTelemetryFields requests	Each supported Region: 10	No	Lists fields available for telemetry queries.
Rate of ListTelemetryQuerySessions requests	Each supported Region: 10	No	Lists telemetry query sessions.
Rate of ListViews requests	Each supported Region: 10	No	Lists the views in the callers account and region.
Rate of PutIntelligenceConfiguration requests	Each supported Region: 10	No	Creates or updates the intelligence configuration for the calling account.
Rate of SearchPrincipals requests	Each supported Region: 50	No	Searches Identity Center for users and groups matching a query within a domain.
Rate of StartTelemetryQuery requests	Each supported Region: 10	No	Starts a telemetry query within a session.
Rate of StartTelemetryQuerySession requests	Each supported Region: 10	No	Starts a new telemetry query session.
Rate of StopTelemetryQuery requests	Each supported Region: 10	No	Stops a running telemetry query.
Rate of StopTelemetryQuerySession requests	Each supported Region: 10	No	Stops a telemetry query session.

Name	Default	Adjustable	Description
Rate of UpdateAccessProfile requests	Each supported Region: 5	No	Updates the name and/or description of an Access Profile.
Rate of UpdateAlert requests	Each supported Region: 5	No	Updates an existing alert.
Rate of UpdateDomain requests	Each supported Region: 5	No	Updates a domains authentication provider configuration.
Rate of UpdateDomainForOrganization requests	Each supported Region: 5	No	Updates an organization-scoped domain. Must target the domains home region.
Rate of UpdateIntegration requests	Each supported Region: 10	No	Updates an existing integration, identified by its id, ARN, or name.
Rate of UpdateOmniDashboard requests	Each supported Region: 5	No	Updates an existing dashboard within a space.
Rate of UpdateSpace requests	Each supported Region: 5	No	Updates an existing Space within a domain.
Rate of UpdateView requests	Each supported Region: 10	No	Updates an existing views definition and/or description.

CloudWatch RUM

CloudWatch RUM (Real-User Monitoring) helps you collect and analyze performance data from real user interactions with your web applications. This section presents the quotas for CloudWatch RUM, allowing you to plan your real-user monitoring strategy within the service limits.

Name	Default	Adjuse	Description
RUM AppMonitors	Each supported Region: 50	Yes	The maximum number of RUM AppMonitors that you can have in this AWS account.
RUM Events per second per AWS Account	Each supported Region: 50	Yes	The maximum number of RUM Events per second that RUM will ingest for this AWS account.

Managing your CloudWatch service quotas

Effective management of CloudWatch service quotas is crucial for maintaining optimal performance and avoiding service disruptions. This section provides guidance on viewing your current quota usage, requesting quota increases when necessary, and best practices for working within CloudWatch service limits.

CloudWatch quotas are integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*.

AWS Management Console

To view CloudWatch service quotas using the console

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.

3. From the **AWS services** list, search for and select the CloudWatch service whose quotas you want to view.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

4. To view additional information about a service quota, such as the description, choose the quota name.

5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the console see the Service Quotas User Guide. To learn more about quota increases, see Requesting a quota increase in the *Service Quotas User Guide*.

AWS CLI

To view CloudWatch service quotas using the AWS CLI

1. Run the following command to view the default CloudWatch quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code ServiceCode
    --output table
```

> **Note**
>
> The following list contains the CloudWatch **ServiceCode** values:
>
> - To see the CloudWatch quotas, use `--service-code monitoring`.
> - To see the Amazon AI Operations quotas, use `--service-code aiops`.
> - To see the CloudWatch Application Signals quotas, use `--service-code application-signals`.
> - To see the Amazon CloudWatch Internet Monitor quotas, use `--service-code internetmonitor`.
> - To see the CloudWatch Network Monitor quotas, use `--service-code networkmonitor`.

- To see the CloudWatch Observability Access Manager quotas, use `--service-code oam`.
- To see the CloudWatch RUM (Real-User Monitoring) quotas, use `--service-code rum`.

2. (Optional) Request a quota increase for a CloudWatch service:

   a. Identify the quota code for the quota you want to increase.

   ```
   aws service-quotas list-service-quotas --service-code ServiceCode
   ```

   b. Note the `QuotaCode` value for the item. For example if you wanted to increase your quota for the **Rate of PutCompositeAlarm requests** that are supported by CloudWatch for your account, you would record the **QuotaCode** `L-8742A250`.

   c. Type the following command using the identified `ServiceCode` and `QuotaCode` values and providing the numeric value for the desired new quota:

   ```
   aws service-quotas request-service-quota-increase \
       --service-code ServiceCode \
       --quota-code QuotaCode \
       --desired-value new-quota-value
   ```

   For more details about quota increases, see the `request-service-quota-increase` command in the AWS CLI Command Reference.

   To work more with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference.

Document history

The following table describes important changes for each release of the *Amazon CloudWatch User Guide*, beginning in June 2018. To receive notifications about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
Updated CloudWatchOmniSpaceAccessPolicy managed policy	CloudWatch added the `cloudwatch:GetProfileRecords` permission to the **CloudWatchOmniSpaceAccessPolicy** managed policy. With this permission, you can query continuous profiling data in a CloudWatch Omni space. For more information, see AWS managed policies for CloudWatch Omni.	November 18, 2026
Network health indicator supports inter-Region paths	The network health indicator (NHI) can now show whether degradation on a path to a destination in another AWS Region is within the AWS network. Supported path types are AWS Transit Gateway inter-Region peering, inter-Region VPC peering, public IP addresses of Amazon EC2, and AWS PrivateLink. For more information about the supported path types, see Network health indicator.	October 28, 2026

Transaction Search supports customer managed keys	You can now encrypt the `aws/spans` log group that stores your spans with a customer managed AWS KMS key. Choose the key when you enable Transaction Search in the console, with the X-Ray API, or with CloudFormation. For more information about the key requirements, see Encrypting spans with a customer managed key.	October 23, 2026

CloudWatchFullAccessV2 policy update	To let you create log groups that are encrypted with your own keys, CloudWatch added the `kms:DescribeKey` permission to the CloudWatchFullAccessV2 managed policy. CloudWatch Logs requires this permission from the caller to create a log group with a customer managed AWS KMS key, or to associate a key with a log group. The key policy must also allow CloudWatch Logs to use the key. You can use this permission only for requests that CloudWatch Logs makes on your behalf, and only for keys in your own account. For example, you need it to encrypt the `aws/spans` log group with a customer managed key when you enable Transaction Search. For more information about policy updates, see CloudWatch updates to AWS managed policies.	October 20, 2026

Updated CloudWatchOmniSpaceAccessPolicy managed policy	CloudWatch added the `identitystore:ListUsers`, `identitystore:ListGroups`, `identitystore:DescribeUser`, `identitystore:DescribeGroup`, `kms:Decrypt`, and `sns:ListTopics` permissions to the **CloudWatchOmniSpaceAccessPolicy** managed policy. With these permissions, you can search for IAM Identity Center users and groups when you grant access to a CloudWatch Omni space, and choose an Amazon SNS topic as an alert notification target. For more information, see AWS managed policies for CloudWatch Omni.	October 14, 2026
CloudWatch OTel enrichment supports filtering enriched vended metrics	CloudWatch OTel enrichment now supports filtering enriched vended metrics by namespace and metric name using `IncludeFilters` and `ExcludeFilters`, and updating filters on a running account with `UpdateOTelEnrichment`. For more information, see Filtering enrichment by namespace and metric.	October 5, 2026

Reorganized telemetry collection content under a new chapter	CloudWatch grouped the telemetry ingestion content under a new *Collect and send telemetry* chapter, which now includes the OpenTelemetry sections, the CloudWatch agent, embedded metric format, CloudWatch Pipelines, and managed Prometheus collectors. For more information, see OpenTelemetry.	October 1, 2026
Amazon CloudWatch pipelines is available in the AWS European Sovereign Cloud	You can now create Amazon CloudWatch pipelines in AWS European Sovereign Cloud (Germany). For more information, see Region availability.	September 17, 2026

Updated CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy managed policy	CloudWatch updated the **CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy** to include additional permissions. `elasticloadbalancing:` permissions describe load balancers, listeners, rules, target groups, and target health. `networkmanager:` permissions describe AWS Cloud WAN core networks, attachments, and routes. `config:` and `iam:` permissions create, delete, and describe a service-linked configuration recorder and create the AWS Config service-linked role. With these permissions, Network Flow Monitor can generate topology snapshots of Elastic Load Balancing and AWS Cloud WAN resources and record resource configuration data through AWS Config. For more information, see CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy.	August 31, 2026

Network Synthetic Monitor extends the network health indicator to AWS Transit Gateway inter-Region peering	The network health indicator (NHI) now covers paths that reach a destination across an AWS Transit Gateway inter-Region peering connection. For more information, see Network health indicator.	August 28, 2026
Updated CloudWatchAutomaticDashboardsAccess managed policy	CloudWatch added the `lambda:GetFunctionConfiguration` and `lambda:ListTags` permissions to the CloudWatchAutomaticDashboardsAccess managed policy to enable automatic dashboards to retrieve Lambda function configuration details and resource tags.	August 6, 2026
Telemetry configuration adds support for Amazon S3 server access logs	CloudWatch telemetry configuration now supports discovering Amazon Simple Storage Service (Amazon S3) buckets and enabling server access log delivery to CloudWatch Logs. You can create enablement rules to automatically configure server access logs for your Amazon S3 buckets across your account or organization.	June 30, 2026

Updated CloudWatchFullAccessV2 policy	CloudWatch added two `iam:PassRole` permissions to CloudWatchFullAccessV2 to support the new `PutLogAlarm` API. The new permissions allow customers to pass a Scheduled Query Execution Role to `logs.amazonaws.com` and a Log Lines Role to `cloudwatch.amazonaws.com` .	June 15, 2026
Internet Monitor publishes local health events to Amazon EventBridge	Internet Monitor now publishes local health events to Amazon EventBridge, in addition to overall (global) health events. Local health events have an impact type of `LOCAL_AVAILABILITY` or `LOCAL_PERFORMANCE` .	May 20, 2026
Updated AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy	CloudWatch updated the **AWSObservabilityAdminTelemetryEnablementServiceRolePolicy** that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the newly supported AWS resources based on telemetry rules. For more information, see AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.	April 30, 2026

Updated CloudWatchSyntheticsFullAccess managed policy	The `cloudwatch:ListMetrics` permission was added to the **CloudWatchSyntheticsFullAccess** policy. Additionally, the `apigateway:GET` permission has been changed from allowing all resources to specific API Gateway resources: REST APIs, REST API stages, REST API stage Swagger exports, and HTTP APIs. For more information, see CloudWatchSyntheticsFullAccess.	March 31, 2026
Updated AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy	CloudWatch updated the **AWSObservabilityAdminTelemetryEnablementServiceRolePolicy** that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the newly supported AWS resources based on telemetry rules. For more information, see AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.	March 31, 2026

Updated AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy	CloudWatch updated the **AWSObservabilityAdminTelemetryEnablementServiceRolePolicy** that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the newly supported AWS resources based on telemetry rules. For more information, see AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.	March 10, 2026
Update to AIOPsAssistantPolicy managed policy	CloudWatch updated the **AIOpsAssistantPolicy** to allow CloudWatch investigations access to additional resources to assist in queries, troubleshooting, and topology mapping. For more information, see AIOpsAssistantPolicy.	February 6, 2026
Update to AIOPsAssistantPolicy managed policy	CloudWatch updated the **AIOpsAssistantPolicy** to allow CloudWatch investigations access to additional resources to assist in queries, troubleshooting, and topology mapping. For more information, see AIOpsAssistantPolicy.	December 17, 2025

Updated CloudWatchNetworkMonitorServiceRolePolicy managed policy with TGW permissions	CloudWatch added the `ec2:DescribeRouteTables` , `ec2:DescribeTransitGatewayAttachments` , `ec2:DescribeTransitGatewayRouteTables` , `ec2:SearchTransitGatewayRoutes` permissions to the CloudWatchNetworkMonitorServiceRolePolicy managed policy to grant permissions for Network Synthetic Monitor to generate the Network Health Indicator values for customers utilizing Transit Gateways.	December 12, 2025
Updated CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy managed policy with VPC Endpoints and vpce service config permissions	CloudWatch added the `ec2:DescribeVpcEndpoints` and `ec2:DescribeVpcEndpointServiceConfigurations` permissions to the CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy managed policy to grant permissions for Network Flow Monitor to generate topology snapshots of VPC endpoints and vpce service config.	December 10, 2025

Updated CloudWatchFullAccessV2 policy	Permissions for observability administration actions were added to CloudWatchFullAccessV2 to allow full access to telemetry pipelines and S3 table integrations.	December 2, 2025
Observability for Amazon Bedrock AgentCore policies	You can now monitor invocation metrics for Amazon Bedrock AgentCore policies using Amazon CloudWatch. For more information, see Gateways.	December 2, 2025
Observability for Amazon Bedrock AgentCore Evaluations	You can now monitor and assess the performance, quality, and reliability of your AI agents using the Evaluations dashboard. For more information, see Agent details - Evaluations.	December 2, 2025
New AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy	CloudWatch updated the **AWSObservabilityAdminTelemetryEnablementServiceRolePolicy** that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for the newly supported AWS resources based on telemetry rules. For more information, see AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.	December 2, 2025

Introduced CloudWatch pipelines	The new pipeline feature of CloudWatch connects data from a catalogue of data sources, so that you can add and configure pipeline processors from a library to parse, enrich, and standardize data. For more information, see CloudWatch pipelines.	December 2, 2025
Introduced CloudWatch pipelines	The new pipeline feature of CloudWatch connects data from a catalogue of data sources, so that you can add and configure pipeline processors from a library to parse, enrich, and standardize data. For more information, see CloudWatch pipelines.	December 2, 2025

New CloudWatch investigations feature	Amazon CloudWatch now offers interactive incident report generation, enabling customers to create comprehensive post-incident analysis reports in minutes. The new capability, available within CloudWatch investigations, automatically gathers and correlates your telemetry data, as well as your input and any actions taken during an investigation, and produces a streamlined incident report. Amazon CloudWatch enhanced incident report generation capabilities with an AI-powered root-cause workflow that guides customers through the "Five Why's" analysis technique. For more information, see Generate incident reports	November 30, 2025
Updated CloudWatchReadOnlyAccess policy	CloudWatch updated CloudWatchReadOnlyAccess to include Resource Explorer permissions to support uninstrumented service functionalities in Application Signals.	November 20, 2025

Updated CloudWatchFullAccessV2 policy	CloudWatch updated CloudWatchFullAccessV2 to include Resource Explorer permissions to support uninstrumented service functionalities in Application Signals.	November 20, 2025
Updated CloudWatchApplicationSignalsServiceRolePolicy policy	CloudWatch updated the policy to include `resource-explorer-2:Search` and `cloudtrail:CreateServiceLinkedChannel` to enable new Application Signals features. For more information, see CloudWatchApplicationSignalsServiceRolePolicy.	November 20, 2025
Updated CloudWatchApplicationSignalsReadOnlyAccess policy	CloudWatch updated CloudWatchApplicationSignalsReadOnlyAccess to include new Application Signals operation and OAM and Resource Explorer permissions to support change events, cross account and uninstrumented service functionalities in Application Signals.	November 20, 2025

Updated CloudWatchApplicationSignalsFullAccess policy	CloudWatch updated CloudWatchApplicationSignalsFullAccess to include OAM and Resource Explorer permissions to support cross account and uninstrumented service functionalities in Application Signals.	November 20, 2025
CloudWatch updated AIOpsConsoleAdminPolicy managed policy with Amazon Q permissions	CloudWatch updated the **AIOpsConsoleAdminPolicy** managed policy to include Amazon Q integration permissions. These permissions enable users to interact with CloudWatch investigations incident reports through Amazon Q's conversational interface. For more information see, AIOpsConsoleAdminPolicy.	November 17, 2025
CloudWatch updated AIOpsOperatorAccess managed policy with Amazon Q permissions	CloudWatch updated the **AIOpsOperatorAccess** managed policy to include Amazon Q integration permissions. These permissions enable users to interact with CloudWatch investigations incident reports through Amazon Q's conversational interface. For more information see, AIOpsOperatorAccess.	November 7, 2025

Updated CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy managed policy with managed prefix lists permissions	CloudWatch added the `ec2:DescribeManagedPrefixLists` and `ec2:GetManagedPrefixListEntries` permissions to the CloudWatchNetworkFlowMonitorTopologyServiceRolePolicy managed policy to grant permissions for Network Flow Monitor to generate topology snapshots of managed prefix lists.	November 6, 2025
Updated CloudWatchReadOnlyAccess Policy with observability administration permissions	CloudWatch updated the **CloudWatchReadOnlyAccess** managed policy to include observability administration permissions. These permissions provide read-only access to telemetry rules, centralization configurations, and resource telemetry data across AWS Organizations. For more information, see CloudWatchReadOnlyAccess.	October 10, 2025

CloudWatch updated AIOpsOperatorAccess managed policy with incident report permissions	CloudWatch updated the **AIOpsOperatorAccess** managed policy to include incident report generation permissions. These permissio ns allow users to create and manage incident reports and work with AI-derived facts in CloudWatch investigations. For more information, see AIOpsOperatorAccess.	October 10, 2025
CloudWatch investigations added incident reports	CloudWatch updated CloudWatch investigations to provide incident reporting . These AI-generated reports capture investigation findings, root causes, timeline events, and recommended corrective actions in a structured format. For more information see, Generate incident reports.	October 10, 2025

CloudWatch added AIOpsAssistantIncidentReportPolicy managed policy to support CloudWatch investigations AI-generated incident reports	CloudWatch added the **AIOpsAssistantIncidentReportPolicy** managed policy to enable CloudWatch investigations to generate incident reports from investigation data. This policy provides permissions to access investigations, create and manage reports, and work with AI-derived facts that form the foundation of incident documentation. For more information, see AIOpsAssistantIncidentReportPolicy.	October 10, 2025
Updated CloudWatchReadOnlyAccess policy with CloudTrail and Service Quotas permissions	CloudWatch updated CloudWatchReadOnlyAccess to include CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals.	October 8, 2025
Updated CloudWatchFullAccessV2 policy	CloudWatch updated CloudWatchFullAccessV2 to include CloudTrail and Service Quotas permissions to support top observations and change indicators functionality within Application Signals.	October 8, 2025

Updated CloudWatchApplicationSignalsReadOnlyAccess policy	CloudWatch updated the `CloudWatchApplicationSignalsReadOnlyAccess` policy to view how resources and services change in your account and view top observations for service anomalies in your account. For more information, see CloudWatchApplicationSignalsReadOnlyAccess.	September 29, 2025
Updated CloudWatchApplicationSignalsFullAccess policy	CloudWatch updated the `CloudWatchApplicationSignalsFullAccess` policy to view how resources and services change in your account and view top observations for service anomalies in your account. For more information, see CloudWatchApplicationSignalsFullAccess.	September 29, 2025
CloudWatch Application Signals added support for dynamic service grouping and filtering, last deployment tracking, and automated audit findings	You can group and filter services with Application Signals' dynamic grouping capabilities track deployment activities through Application Signals' automatic processing of CloudTrail events, and discover critical insights through Application Signals' automated audit findings. For more information, see Application Signals.	September 29, 2025

Update to AIOPsAssistantPolicy managed policy	CloudWatch updated the **AIOpsAssistantPolicy** to allow CloudWatch investigations access to additional resources to assist in queries, troubleshooting, and topology mapping. For more information, see AIOpsAssistantPolicy.	September 24, 2025
New AWSObservabilityAdminLogsCentralizationServiceRolePolicy policy	CloudWatch added the **AWSObservabilityAdminLogsCentralizationServiceRolePolicy** that grants CloudWatch permission to use telemetry data from other AWS accounts that you specify to create CloudWatch log groups, log streams, and log events in your organization's monitoring account. The SLR only provides the assume role permission to allow the CloudWatch service to assume the role in the monitoring account. For more information, see AWSObservabilityAdminLogsCentralizationServiceRolePolicy.	September 17, 2025

CloudWatch added support for Cross-account cross-Region log centralization	CloudWatch added support for the **Cross-account cross-Region log centralization** that enables CloudWatch Logs to gather log data from AWS Organizations member accounts into a central account for analysis. For more information, see Monitor across accounts and Regions.	September 17, 2025
Collect EC2 instance store volume NVMe driver metrics	You can now use the CloudWatch agent to collect AWS NVMe driver metrics for instance store volumes attached to Nitro-based Amazon EC2 instances.	September 16, 2025
Removed AWS OpsWorks metrics and dimensions	AWS OpsWorks reached end-of-life and the service is deprecated.	August 20, 2025
New AWSObservabilityAdminTelemetryEnablementServiceRolePolicy policy	CloudWatch added the **AWSObservabilityAdminTelemetryEnablementServiceRolePolicy** that grants CloudWatch the permissions necessary to enable and manage telemetry configurations for AWS resources based on telemetry rules. For more information, see AWSObservabilityAdminTelemetryEnablementServiceRolePolicy.	July 17, 2025

Generative AI observability	With CloudWatch, you can observe generative AI workloads, including Amazon Bedrock AgentCore agents, and gain insights into AI performance, health, and accuracy. For more information, see Generative AI observability.	July 16, 2025
AIOpsReadOnlyAccess policy updated	CloudWatch updated the **AIOpsReadOnlyAccess** policy to permit validation of investigation groups across accounts. For more informati on, see AIOpsReadOnlyAccess policy.	June 23, 2025
Updated AIOpsCons oleAdminPolicy policy	CloudWatch updated the AIOpsConsoleAdminPolicy policy to permit validation of investigation groups across accounts. For more informati on, see AIOpsConsoleAdminP olicy policy.	June 13, 2025
Updated AIOpsAssistantPolicy policy	CloudWatch updated the AIOpsAssistantPolicy policy to provide additional IAM permissions. For more information, see AIOpsCons oleAdminPolicy policy.	June 13, 2025

Update managed policy	CloudWatch updated the **AIOpsOperatorAccess** policy to permit validation of investigation groups across accounts. For more information, see AIOpsOperatorAccess.	June 13, 2025
AmazonCloudWatchRUMReadOnlyAccess updated	The `synthetics:describeCanaries`, `synthetics:describeCanariesLastRun`, `cloudwatch:GetMetricData`, `cloudwatch:DescribeAlarms`, `logs:DescribeLogGroups`, `xray:GetTraceSummaries`, and `rum:ListTagsForResources` permissions was added to the CloudWatch RUM `AmazonCloudWatchRUMReadOnlyAccess` policy.	June 12, 2025
Added functionality	You can now monitor Amazon Aurora PostgreSQL Limitless Databases at the instance level from the Database Instance Dashboard. For more information, see Monitoring Aurora Limitless databases with Database Insights.	May 22, 2025

Added functionality	You can now enable Application Signals for your applications on an existing Amazon EKS cluster using CloudWatch Observability add-on Auto monitor advanced configuration. For more information, see Enable Application Signals on an Amazon EKS cluster using the CloudWatch Observability add-on advanced configuration.	May 19, 2025
CloudWatch Network Monitoring releases multiple account support for Network Flow Monitor	Network Flow Monitor, a feature of CloudWatch Network Monitoring, now supports monitoring network performance metrics for network flows for resources in multiple accounts. For more information, see Initialize Network Flow Monitor for multi-account monitoring.	May 6, 2025
AmazonCloudWatchRUMReadOnlyAccess updated	The `rum:GetResourcePolicy` permission was added so that CloudWatch RUM can view the resource policy attached to the RUM application monitor.	April 28, 2025

Updated AIOpsAssistantPolicy policy	CloudWatch added permissions to the **AIOpsAssistantPolicy** policy, which Amazon Q Developer operational investigations uses to find information during investigations.	April 11, 2025
CloudWatch Synthetics deprecates five runtime metrics on October 1, 2025	CloudWatch Synthetics metrics `syn-nodejs-puppeteer-7.0` , `syn-nodejs-puppeteer-6.2` , `syn-nodejs-puppeteer-5.2` , `syn-python-selenium-3.0` , and `syn-python-selenium-2.1` will be deprecated on October 1, 2025.	April 4, 2025
Updated CloudWatchApplicationSignalsServiceRolePolicy policy	CloudWatch updated the policy to exclude time windows from impacting the SLO attainment rate, error budget, and burn rate metrics. CloudWatch can retrieve exclusion windows on behalf of you. For more information, see CloudWatchApplicationSignalsServiceRolePolicy.	March 13, 2025

Time window exclusions	SLOs adds support for time window exclusions which is a block of time with a defined start and end date. This time period is excluded from the SLO's performance metrics and you can schedule one-time or recurring time exclusions windows. For more information, see Service level objectives (SLOs).	March 13, 2025
New AIOpsReadOnlyAccess policy	CloudWatch added the **AIOpsReadOnlyAccess** policy, which grants read-only permissions for Amazon AI Operations and other related services.	December 3, 2024
New AIOpsOperatorAccess policy	CloudWatch added the **AIOpsOperatorAccess** policy, which grants users access to Amazon AI Operations actions, and to additiona l AWS actions that are necessary for accessing investigation events.	December 3, 2024

New AIOpsConsoleAdminPolicy policy	CloudWatch added the **AIOpsConsoleAdminPolicy** policy, which grants users full administrative access for managing Amazon Q investigations, including the management of trusted identity propagation, and the management of IAM Identity Center and organizational access.	December 3, 2024
New AIOpsAssistantPolicy policy	CloudWatch added the **AIOpsAssistantBasicAccess** policy, which you can assign to the Amazon AI Operations assistant to enable the assistant to have basic functionality.	December 3, 2024
Amazon Q Developer operational investigations feature in Preview	CloudWatch released a Preview of Amazon Q Developer operational investigations. This feature is a generative AI-powered assistant that can help you respond to incidents in your system. It uses generative AI to scan your system's telemetry and quickly surface suggestions that might be related to your issue, to help you troubleshoot operational issues faster.	December 3, 2024

CloudWatchFullAccessV2 and CloudWatchFullAccess policies updated	CloudWatch updated both **CloudWatchFullAccessV2** and **CloudWatchFullAccess**. Permissions for Amazon OpenSearch Service were added to to enable CloudWatch Logs integration with OpenSearch Service for some features.	December 1, 2024
Telemetry config feature generally available	CloudWatch added the telemetry config feature to discover and understand the state of telemetry configuration for your AWS resources.	November 26, 2024
New AWSServiceRoleForObservabilityAdmin service-linked role	CloudWatch added the new AWSServiceRoleForObservabilityAdmin service-linked role and AWSObservabilityAdminServiceRolePolicy policy.	November 26, 2024
CloudWatch supports exploring related telemetry	CloudWatch supports exploring related telemetry data across interconnected resources.	November 22, 2024

Updated CloudWatchSyntheticsFullAccess IAM policy	CloudWatch updated the **CloudWatchSyntheticsFullAccess** policy to grant CloudWatch Synthetics access to get AWS Lambda functions , and perform actions on logs available in CloudWatch Logs. Additionally, Lambda layer version actions apply to all CloudWatch Synthetics layer ARNs.	November 21, 2024
New CloudWatch Synthetics support for Node.js with the Playwright runtime	CloudWatch Synthetics adds support for Node.js canary scripts that use the Playwright runtime.	November 21, 2024
CloudWatchReadOnlyAccess policy updated	The scope of the policy was changed to include the `xray:List *`, `xray:StartTraceRetrieval` , and `xray:CancelTraceRetrieval` permissions. This change allows CloudWatch to access new X-Ray read APIs. For more infomration, see CloudWatchReadOnlyAccess.	November 21, 2024

CloudWatch Transaction Search is generally available	CloudWatch releases Transaction Search, a search and analytics experience. Transaction Search provides visibility into application transaction issues. Explore interactions between application components to establish a root cause. Understand your application transactions and their impact on customers. Search transactions using terms like customer name or order. For more information, see CloudWatch Transaction Search.	November 21, 2024
Application Signals supports Lambda functions	You can now enable Application Signals for Lambda functions.	November 21, 2024
OpenTelemetry	You can use OpenTelemetry to directly send traces to an OpenTelemetry Protocol (OTLP) endpoint, and get out-of-the box application performance monitoring experiences in CloudWatch Application Signals.	November 20, 2024
Application Signals collects runtime metrics	Application Signals now collects OpenTelemetry-compatible runtime metrics from your Java and Python applications.	November 19, 2024

Application Signals collects runtime metrics	Application Signals now collects OpenTelemetry-compatible runtime metrics from your Java and Python applications.	November 19, 2024
Internet traffic monitoring for VPCs in AWS Local Zones	Internet Monitor now supports internet traffic monitoring for VPCs deployed in AWS Local Zones	November 18, 2024
Application Signals support for Node.js applications is generally available	CloudWatch supports enabling Node.js applications for Application Signals.	November 15, 2024
New CloudWatchInternetMonitorReadOnlyAccess IAM policy	CloudWatch added a **CloudWatchInternetMonitorReadOnlyAccess** policy, to grant read-only access to actions and resources available in the CloudWatch console for Internet Monitor.	November 14, 2024
CloudWatch added observability solutions	CloudWatch observability solutions offer a catalog of readily available configurations to help you quickly implement monitoring for various AWS services and common workloads, such as Java Virtual Machines (JVM), Apache Kafka, Apache Tomcat, and NGINX.	November 13, 2024

CloudWatch added burn rate monitoring for Application Signals SLOs	CloudWatch added burn rate monitoring to Application Signals SLOs. A burn rate is a metric that indicates how fast the service is consuming the error budget, relative to the attainment goal of the SLO.	November 13, 2024
New Internet Monitor latency reduction suggestions through Route 53 routing updates	Amazon CloudWatch Internet Monitor now provides actionable suggestions to help you optimize your Route 53 IP-based routing configurations. Easily identify the optimal AWS Regions for routing your end user traffic, and then configure your Route 53 IP-based routing based on the recommend ations.	November 1, 2024
New CloudWatchInternet MonitorFullAccess IAM policy	CloudWatch added a **CloudWatchInternet MonitorFullAccess** policy, to grant full access to actions and resources available in the CloudWatch console for Internet Monitor.	October 23, 2024
End of support notice	End of support notice: On October 16, 2025, AWS will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.	October 17, 2024

End of support notice	End of support notice: On October 16, 2025, AWS will discontinue support for CloudWatch Evidently. After October 16, 2025, you will no longer be able to access the Evidently console or Evidently resources.	October 17, 2024
New CloudWatchLambdaApplicationSignalsExecutionRolePolicy IAM policy	CloudWatch added a new policy named **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024
New CloudWatchLambdaApplicationSignalsExecutionRolePolicy IAM policy	CloudWatch added a new policy named **CloudWatchLambdaApplicationSignalsExecutionRolePolicy**. This policy is used when CloudWatch Application Signals is enabled for Lambda workloads. It enables write access to X-Ray and the log group used by CloudWatch Application Signals.	October 16, 2024

CloudWatchSyntheticsFullAccess policy updated	Permissions were added to the **CloudWatchSyntheticsFullAccess** IAM policy. The `lambda:ListTags` , `lambda:TagResource` , and `lambda:UntagResource` permissions were added so that when you apply or change tags on a canary, you can choose to have Synthetics also apply those same tags or changes to the Lambda function that the canary uses.	October 11, 2024
Application Signals supports request-based SLOs	Application Signals adds support for request-based service level objects (SLOs). For more information, see Service level objectives (SLOs).	September 6, 2024
Internet Monitor refreshed dashboard and latency improvement suggestions	Amazon CloudWatch Internet Monitor has launched an updated console experience, including new features for visualizing configuration changes that can help you reduce latency for your application.	August 12, 2024
Application Signals previewing support for .NET applications	CloudWatch Application Signals has added support for .NET applications, with ADOT Instrumentation for .NET on supported platforms. This feature is in preview release.	August 5, 2024

New CloudWatch Application Insights service linked role permission added	Allows CloudWatch Application Insights to enable and disable termination protection on CloudFormation stacks.	July 25, 2024
CloudWatch Metrics Insights support for natural language query generation is generally available.	CloudWatch Metrics Insights supports natural language query to generate and update queries. For more information, see Use natural language to generate and update CloudWatch Metric Insights queries.	June 10, 2024
CloudWatch Application Signals is generally available	CloudWatch Application Signals is now generally available. Use Application Signals to instrument your applications on AWS so that you can monitor current application health, create service level objectives (SLOs), and track long-term application performance against your business objectives. For more information, see Application Signals.	June 10, 2024
New CloudWatchApplicationSignalsReadOnlyAccess IAM policy	CloudWatch added a **CloudWatchApplicationSignalsReadOnlyAccess** policy to add read only actions and resources available in the CloudWatch console for Application Signals.	June 7, 2024

New CloudWatchApplicationSignalsFullAccess policy	CloudWatch added a **CloudWatchApplicationSignalsFullAccess** policy to add actions and resources available in the CloudWatch console for Application Signals.	June 7, 2024
CloudTrail now captures API activities related to CloudWatch data plane operations.	CloudTrail now logs events in CloudTrail for **GetMetricData** and **GetMetricWidgetImage** API activities.	June 6, 2024
CloudWatch Application Signals application map supports canary, RUM clients and AWS service dependency groupings.	The Application Signals preview release has added default groupings in the application map for canaries, RUM clients, and AWS service dependencies of the same type. This change reduces the number of icons in the application map default view to make it easier to view and navigate.	May 21, 2024

CloudWatchReadOnlyAccess policy updated	CloudWatch changed the scope of a permission in **CloudWatchReadOnlyAccess**. The scope of the policy added the `application-signals:BatchGet*` , `application-signals:Get*` , and `application-signals:List*` actions so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services. CloudWatch also added an `iam:GetRole` action so that users can check if Application Signals is set up.	May 17, 2024
CloudWatchFullAccessV2 policy updated	CloudWatch changed the scope of a permission in **CloudWatchFullAccessV2**. The scope of the policy added the `application-signals:*` so that users can use CloudWatch Application Signals to view, investigate, and diagnose issues with the health of their services.	May 17, 2024
Lambda Insights supports AWS GovCloud (US-East) and AWS GovCloud (US-West)	CloudWatch Lambda Insights has added support for the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions.	April 29, 2024

CloudWatch cross-account observability supports resource filters	You can now create filters to specify which metric namespaces and log groups are shared from the source account to the monitoring account, when you create the link between the accounts.	April 26, 2024
CloudWatch Application Signals updates	The Application Signals preview release has added three features. Applicati on Signals now supports Python applications. It offers a simpler enablement process for applications on Amazon EKS architectures. And it includes new configurations that you can use to manage the cardinality of metrics that are collected.	April 26, 2024
CloudWatch Container Insights with enhanced observability for Amazon EKS can collect AWS Elastic Fabric Adapter (EFA) metrics	You can now use CloudWatc h Container Insights with enhanced observability for Amazon EKS collect AWS Elastic Fabric Adapter (EFA) metrics from Amazon EKS clusters.	April 23, 2024

Updated IAM policy	CloudWatch updated the **CloudWatchApplicationSignalsServiceRolePolicy** policy. The scoping of the `logs:StartQuery` and `logs:GetQueryResults` permissions in this policy was changed to add `arn:aws:logs:*:*:log-group:/aws/appsignals/*:*` and `"arn:aws:logs:*:*:log-group:/aws/application-signals/data:*"` to enable Application Signals on more architecures. This policy is attached to the **AWSServiceRoleForCloudWatchApplicationSignals** service-linked role.	April 18, 2024
Internet Monitor provides a global internet weather map to authenticated AWS customers	Amazon CloudWatch Internet Monitor now displays a global internet weather map that is available in the console to all authenticated AWS customers. To view the map, in the Amazon CloudWatch console, navigate to Internet Monitor.	April 16, 2024
CloudWatch Container Insights with enhanced observability for Amazon EKS can collect AWS Neuron metrics	You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect AWS Neuron metrics from Amazon EKS clusters.	April 16, 2024

CloudWatch Application Signals adds a **Service overview** tab and more metrics to aid in diagnostics	A new **Service overview** tab displays an overview of your service, including number of operations, dependencies, synthetics, and client pages. The tab shows key metrics for your entire service, and top operations and dependencies. You can also now view X-Ray traces that are correlated with issues including faults, errors, and latency problems.	April 16, 2024
CloudWatch Container Insights with enhanced observability for Amazon EKS adds support for Windows	You can now use CloudWatch Container Insights with enhanced observability for Amazon EKS collect metrics from Windows worker nodes on Amazon EKS clusters.	April 10, 2024
CloudWatchApplicationSignalsServiceRolePolicy IAM policy updated	CloudWatch changed the scope of a permission in **CloudWatchApplicationSignalsServiceRolePolicy**. The scope of the `cloudwatch:GetMetricData` permission was changed to * so that Application Signals can retrieve metrics from sources in linked accounts.	April 8, 2024
Amazon CloudWatch Internet Monitor now supports cross-account observability	You can now use Internet Monitor cross-account observability to monitor your applications that span multiple AWS accounts within a single AWS Region.	March 29, 2024

CloudWatchAgentServerPolicy and **CloudWatchAgentAdminPolicy** policies updated	CloudWatch added permissions to both the `CloudWatchAgentServerPolicy` and `CloudWatchAgentAdminPolicy` policies to allow the CloudWatch agent to publish X-Ray traces and modify log group retention periods. In both policies, the `xray:PutTraceSegments`, `xray:PutTelemetryRecords`, `xray:GetSamplingRules`, `xray:GetSamplingTargets`, `xray:GetSamplingStatisticSummaries` and `logs:PutRetentionPolicy` permissions were added	February 12, 2024

New service linked role and IAM policy for CloudWatch Network Monitor	CloudWatch added a new service-linked role, called **AWSServiceRoleForNetworkMonitor**. CloudWatch added this new service-linked role to allow you to create monitors to fetch network metrics between source subnets and destination IP addresses. The new **CloudWatchNetworkMonitorServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch network metrics on your behalf.	December 22, 2023
CloudWatch releases Amazon CloudWatch Network Monitor	CloudWatch released a new feature, Amazon CloudWatch Network Monitor. This is a new active network monitoring service that identifies if a network issues exists within the AWS network or your own company network.	December 22, 2023

CloudWatchReadOnlyAccess policy updated	CloudWatch added existing read-only permissions for CloudWatch Synthetics, X-Ray, and CloudWatch RUM and new read-only permissions for CloudWatch Application Signals to **CloudWatchReadOnlyAccess** so that users with this policy can triage and diagnose service health issues as reported by CloudWatch Application Signals. The `cloudwatch:GenerateQuery` permission was added so that users with this policy can generate a CloudWatch Metrics Insights query string from a natural language prompt.	December 5, 2023
CloudWatchFullAccessV2 policy updated	CloudWatch added existing permissions to **CloudWatchFullAccessV2** for CloudWatch Synthetics, X-Ray, and CloudWatch RUM, and added new permissions for CloudWatch Application Signals so that users with this policy can fully manage Application Signals to triage and diagnose issues with service health.	December 5, 2023

New service-linked role and new IAM policy	CloudWatch added a new service-linked role, called **AWSServiceRoleForCloudWatchApplicationSignals**. CloudWatch added this new service-linked role to allow CloudWatch Application Signals to collect CloudWatch Logs data, X-Ray trace data, CloudWatch metrics data, and tagging data from applications that you have enabled for CloudWatch Application Signals. The new **CloudWatchApplicationSignalsServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch Application Signals to collect monitoring and tagging data from other relevant AWS services.	November 30, 2023

CloudWatch launches Preview release of Application Signals	CloudWatch Application Signals is in Preview. Use Application Signals to instrument your applicati ons on AWS so that you can monitor current applicati on health, create service level objectives (SLOs), and track long-term application performance against your business objectives. For more information, see Application Signals.	November 30, 2023
CloudWatch adds support for querying other data sources	You can use CloudWatch to query, visualize, and create alarms for metrics from other data sources For more information, see Querying metrics from other data sources.	November 26, 2023
CloudWatch Metrics Insights supports natural language query generation	CloudWatch Metrics Insights supports natural language query to generate and update queries. For more informati on, see Use natural language to generate and update CloudWatch Metric Insights queries.	November 26, 2023

CloudWatch releases Container Insights with enhanced observability for Amazon EKS	CloudWatch released a new version of Container Insights. This version supports enhanced observability for Amazon EKS clusters and can collect more detailed metrics from clusters running Amazon EKS. After installat ion, it automatically collects detailed infrastructure telemetry and container logs for your Amazon EKS clusters. You can then use curated, immediately usable dashboards to drill down into application and infrastructure telemetry.	November 6, 2023
CloudWatch metric streams adds quick partner setup	CloudWatch metric streams now provides a quick partner setup option, which you can use to quickly set up a metric stream to some third-party providers.	October 17, 2023
CloudWatch releases alarm recommendations	CloudWatch Synthetics now provides alarm recommend ations for metrics from other AWS services. These recommendations can help you identify the metrics that you should set alarms for to follow best practices for monitoring these services.	October 16, 2023

CloudWatch Synthetics releases runtime syn-nodejs-puppeteer-6.0	CloudWatch Synthetics released runtime `syn-nodejs-puppeteer-6.0` .	September 26, 2023
Adds Amazon CloudWatch Application Insights support for cross-account applications	You can now share CloudWatch Application Insights applications across account boundaries.	September 26, 2023
New service-linked role and new IAM policy	CloudWatch added a new service-linked role, called **AWSServiceRoleForCloudWatchMetrics_DbPerfInsights**. CloudWatch added this new service-linked role to allow CloudWatch to fetch Performance Insights metrics for alarming, anomaly detection, and snapshotting. The new **AWSServiceRoleForCloudWatchMetrics_DbPerfInsightsServiceRolePolicy** IAM policy is attached to this role, and the policy grants permission to CloudWatch to fetch Performance Insights metrics on your behalf.	September 20, 2023
Adds new metric math function	CloudWatch added a new metric math function, `DB_PERF_INSIGHTS` , that you can use to fetch Performance Insights metrics from AWS database services for alarming, anomaly detection, and snapshotting.	September 20, 2023

CloudWatchReadOnlyAccess policy updated	CloudWatch added the `application-autoscaling:DescribeScalingPolicies` permission to **CloudWatchReadOnlyAccess** so that users with this policy can access information about Application Auto Scaling policies.	September 14, 2023
CloudWatch agent added support for AL2023	The CloudWatch agent supports AL2023.	August 8, 2023
New managed IAM policy, CloudWatchFullAccessV2	CloudWatch added a new policy **CloudWatchFullAccessV2**. This policy grants full access to CloudWatch actions and resources while better scoping the permissions granted to other services such as Amazon SNS and Amazon EC2 Auto Scaling.	August 1, 2023
Updated service linked role for Amazon CloudWatch Internet Monitor – update to an existing policy	Adds new permissions, `elasticloadbalancing:DescribeLoadBalancers` and `ec2:DescribeNetworkInterfaces`, to the service linked role for Internet Monitor, to support monitoring traffic for specific Network Load Balancer resources.	July 25, 2023

Added support for Network Load Balancer resources in Amazon CloudWatch Internet Monitor	Adds support for creating a monitor in Internet Monitor with specific Network Load Balancer resources, to provide more granular levels of observability for your application.	July 25, 2023
Dashboard variables feature	CloudWatch released *dashboard variables*, which you can use to create flexible dashboards that can quickly display different contents depending on how you set one input field within the dashboard. For example, you can create a dashboard that can quickly switch between different Lambda functions or Amazon EC2 instance IDs, or one that can switch to different AWS Regions. For more information, see Create flexible dashboards with dashboard variables.	June 28, 2023
Internet Monitor now supports customizing the threshold for health events	Internet Monitor added the ability to customize the threshold for when a global performance score or availabil ity score triggers a heath event. For more informati on, see Tracking real-time performance and availabil ity in Amazon CloudWatch Internet Monitor.	June 26, 2023

Internet Monitor now supports all commercial Regions	Internet Monitor added seven new AWS Regions and now supports all commercial Regions.	June 19, 2023
New Lambda Insights extension versions	CloudWatch added the 1.0.229.0 version of the Lambda Insights extension for both x86-64 platforms and ARM64 platforms. For more information, see Available versions of the Lambda Insights extension.	June 12, 2023
CloudWatchReadOnlyAccess policy updated	CloudWatch added permissions to **CloudWatchReadOnlyAccess**. The `logs:StartLiveTail` and `logs:StopLiveTail` permissions were added so that users with this policy can use the console to start and stop CloudWatch Logs live tail sessions. For more information, see Use live tail to view logs in near real time.	June 6, 2023

CloudWatch RUM adds support for custom metrics	You can use CloudWatch RUM app monitors to create custom metrics and send them to CloudWatch and CloudWatch Evidently. This feature includes an update to the **AmazonCloudWatchRUMServiceRolePolicy** managed IAM policy. In that policy, a condition key was changed so that CloudWatch RUM can send custom metrics to custom metric namespaces.	February 9, 2023
New and updated managed polices for CloudWatch	To support CloudWatch cross-account observability, the `CloudWatchFullAccess` and `CloudWatchReadOnlyAccess` policies have been updated, and the following new managed policies have been added: `CloudWatchCrossAccountSharingConfiguration`, `OAMFullAccess`, and `OAMReadOnlyAccess`. For more information, see CloudWatch updates to AWS managed policies.	February 7, 2023
CloudWatch Application Insights service-linked role policy updates — update to an existing policy.	CloudWatch Application Insights updated an existing AWS service-linked role policy.	December 19, 2022

Amazon CloudWatch Application Insights support for containerized applications and microservices from the Container Insights console.	You can display CloudWatch Application Insights detected problems for Amazon ECS and Amazon EKS on your Container Insights dashboard.	November 17, 2021
Amazon CloudWatch Application Insights monitoring for SAP HANA databases.	You can monitor SAP HANA databases with Application Insights.	November 15, 2021
Amazon CloudWatch Application Insights support for monitoring all resources in an account.	You can onboard and monitor all resources in an account.	September 15, 2021
Amazon CloudWatch Application Insights support for Amazon FSx.	You can monitor metrics retrieved from Amazon FSx.	August 31, 2021
SDK Metrics is no longer supported.	CloudWatch SDK Metrics is no longer supported.	August 25, 2021
Amazon CloudWatch Application Insights support for setting up container monitoring.	You can monitor containers using best practices with Amazon CloudWatch Application Insights.	May 18, 2021
Metric streams is generally available	You can use metric streams to continually stream CloudWatch metrics to a destination of your choice. For more information, see Metric streams in the *Amazon CloudWatch User Guide*.	March 31, 2021

Amazon CloudWatch Application Insights monitoring for Oracle databases on Amazon RDS and Amazon EC2.	You can monitor metrics and logs retrieved from Oracle with Amazon CloudWatch Application Insights.	January 16, 2021
Lambda Insights is generally available	CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. For more information, see Using Lambda Insights in the *Amazon CloudWatch User Guide*.	December 3, 2020
Amazon CloudWatch Application Insights monitoring for Prometheus JMX exporter metrics.	You can monitor metrics retrieved from Prometheus JMX exporter with Amazon CloudWatch Application Insights.	November 20, 2020
CloudWatch Synthetics releases new runtime version	CloudWatch Synthetics has released a new runtime version. For more information, see Canary Runtime Versions in the *Amazon CloudWatch User Guide*.	September 11, 2020
Amazon CloudWatch Application Insights monitoring for Postgre SQL on Amazon RDS and Amazon EC2.	You can monitor applications built with PostgreSQL running on Amazon RDS or Amazon EC2.	September 11, 2020

CloudWatch supports dashboard sharing	You can now share CloudWatch dashboards with people outside of your organization and AWS account. For more information, see Sharing CloudWatch Dashboards in the *Amazon CloudWatch User Guide*.	September 10, 2020
Set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights	You can use the documentation tutorial to help you to set up monitors for .NET applications using SQL Server on the backend with CloudWatch Application Insights.	August 19, 2020
CloudFormation support for Amazon CloudWatch Application Insights applications.	You can add CloudWatch Application Insights monitoring, including key metrics and telemetry, to your application, database, and web server, directly from CloudFormation templates.	July 30, 2020
Amazon CloudWatch Application Insights monitoring for Aurora for MySQL database clusters.	You can monitor Aurora for MySQL database clusters (RDS Aurora) with Amazon CloudWatch Application Insights.	July 2, 2020

CloudWatch Contributor Insights general availability	CloudWatch Contributor Insights is now generally available. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the *Amazon CloudWatch User Guide*.	April 2, 2020
CloudWatch Synthetics public preview	CloudWatch Synthetics is now in public preview. It enables you to create canaries to monitor your endpoints and APIs. For more information, see Using Canaries in the *Amazon CloudWatch User Guide*.	November 25, 2019

CloudWatch Contributor Insights public preview	CloudWatch Contributor Insights is now in public preview. It enables you to analyze log data and create time series that display contributor data. You can see metrics about the top-N contributors, the total number of unique contributors, and their usage. For more information, see Using Contributor Insights to Analyze High-Cardinality Data in the *Amazon CloudWatch User Guide*.	November 25, 2019
CloudWatch launches ServiceLens feature	ServiceLens ehances the observability of your services and applications by enabling you to integrate traces, metrics, logs, and alarms into one place. ServiceLens integrates CloudWatch with AWS X-Ray to provide an end-to-end view of your application.	November 21, 2019

Use CloudWatch to proactively manage your AWS service quotas	You can use CloudWatch to proactively manage your AWS service quotas. CloudWatch usage metrics provide visibility into your account's usage of resources and API operations. For more information, see Service Quotas Integration and Usage Metrics in the *Amazon CloudWatch User Guide*.	November 19, 2019
CloudWatch sends events when alarms change state	CloudWatch now sends an event to Amazon EventBridge when any CloudWatch alarm changes state. For more information, see Alarm Events and EventBridge in the *Amazon CloudWatch User Guide*.	October 8, 2019
Container Insights	CloudWatch Container Insights is now generally available. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Using Container Insights in the *Amazon CloudWatch User Guide*.	August 30, 2019

Updates for Container Insights preview metrics on Amazon EKS and Kubernetes	The Container Insights on Amazon EKS and Kubernetes public preview has been updated. InstanceId is now included as a dimension to the cluster EC2 instances. This allows alarms that have been created on these metrics to trigger the following EC2 actions: Stop, Terminate, Reboot, or Recover. Additionally, pod and service metrics are now reported by Kubernetes namespace to simplify the monitoring and alarming on metrics by namespace.	August 19, 2019
Updates for AWS Systems Manager OpsCenter integration	Updates on how CloudWatch Application Insights integrates with Systems Manager OpsCenter.	August 7, 2019
CloudWatch usage metrics	CloudWatch usage metrics help you track the usage of your CloudWatch resources and stay within your service limits. For more information, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch-Usage-Metrics.html.	August 6, 2019

CloudWatch Container Insights public preview	CloudWatch Container Insights is now in public preview. It enables you to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. For more information, see Using Container Insights in the *Amazon CloudWatch User Guide*.	July 9, 2019
CloudWatch Anomaly Detection public preview	CloudWatch anomaly detection is now in public preview. CloudWatch applies machine-learning algorithms to a metric's past data to create a model of the metric's expected values. You can use this model for visualization and for setting alarms. For more information, see Using CloudWatch Anomaly Detection in the *Amazon CloudWatch User Guide*.	July 9, 2019
CloudWatch Application Insights for .NET and SQL Server	CloudWatch Application Insights for .NET and SQL Server facilitates observability for .NET and SQL Server applications. It can help you set up the best monitors for your application resources to continuously analyze data for signs of problems with your applications.	June 21, 2019

CloudWatch agent section re-organized	The CloudWatch agent documentation has been rewritten to improve clarity, especially for customers using the command line to install and configure the agent. For more information, see Collecting Metrics and Logs from Amazon EC2 Instances and On-Premises Servers with the CloudWatch Agent in the *Amazon CloudWatch User Guide*.	March 28, 2019
SEARCH function added to metric math expressions	You can now use a SEARCH function in metric math expressions. This enables you to create dashboards that update automatically as new resources are created that match the search query. For more information, see Using Search Expressions in Graphs in the *Amazon CloudWatch User Guide*.	March 21, 2019
AWS SDK Metrics for Enterprise Support	SDK Metrics helps you assess the health of your AWS services and diagnose latency caused by reaching your account usage limits or by a service outage. For more information, see Monitor Applications Using AWS SDK Metrics in the *Amazon CloudWatch User Guide*.	December 11, 2018

Alarms on math expressions	CloudWatch supports creating alarms based on metric math expressions. For more information, see Alarms on Math Expressions in the *Amazon CloudWatch User Guide*.	November 20, 2018
New CloudWatch console home page	Amazon has created a new home page in the CloudWatch console, which automatically displays key metrics and alarms for all the AWS services you are using. For more information, see Getting Started with Amazon CloudWatch in the *Amazon CloudWatch User Guide*.	November 19, 2018
CloudFormation templates for the CloudWatch Agent	Amazon has uploaded CloudFormation templates that you can use to install and update the CloudWatch agent. For more information, see Install the CloudWatch Agent on New Instances Using CloudFormation in the *Amazon CloudWatch User Guide*.	November 9, 2018

Enhancements to the CloudWatch Agent	The CloudWatch agent has been updated to work with both the StatsD and collectd protocols. It also has improved cross-account support. For more informati on, see Retrieve Custom Metrics with StatsD, Retrieve Custom Metrics with collectd, and Sending Metrics and Logs to a Different AWS Account in the *Amazon CloudWatch User Guide*.	September 28, 2018
Support for Amazon VPC endpoints	You can now establish a private connection between your VPC and CloudWatc h. For more information, see Using CloudWatch with Interface VPC Endpoints in the *Amazon CloudWatch User Guide*.	June 28, 2018

The following table describes important changes to the *Amazon CloudWatch User Guide* before June 2018.

Change	Description	Release date
Metric math	You can now perform math expressions on CloudWatch metrics, producing new time series that you can add to graphs on your dashboard. For more information, see Math expressions with metrics.	April 4, 2018
"M out of N" alarms	You can now configure an alarm to trigger based on "M out of N" datapoints in any alarm evaluatio	December 8, 2017

Change	Description	Release date
	n interval. For more information, see Alarm evaluation.	
CloudWatch agent	A new unified CloudWatch agent was released. You can use the unified multi-platform agent to collect custom both system metrics and log files from Amazon EC2 instances and on-premises servers. The new agent supports both Windows and Linux and enables customization of metrics collected, including sub-resource metrics such as per-CPU core. For more information, see Collect metrics, logs, and traces using the CloudWatch agent.	September 7, 2017
NAT gateway metrics	Added metrics for Amazon VPC NAT gateway.	September 7, 2017
High-resolution metrics	You can now optionally set up custom metrics as high-resolution metrics, with a granularity of as low as one second. For more information, see High-resolution metrics.	July 26, 2017
Dashboard APIs	You can now create, modify, and delete dashboards using APIs and the AWS CLI. For more information, see Creating a customized CloudWatch dashboard.	July 6, 2017
Direct Connect metrics	Added metrics for Direct Connect.	June 29, 2017
Amazon VPC VPN metrics	Added metrics for Amazon VPC VPN.	May 15, 2017
WorkSpaces Applications metrics	Added metrics for WorkSpaces Applications.	March 8, 2017

Change	Description	Release date
CloudWatch console color picker	You can now choose the color for each metric on your dashboard widgets. For more information, see Editing a graph on a CloudWatch dashboard.	February 27, 2017
Alarms on dashboards	Alarms can now be added to dashboards. For more information, see Adding an alarm to a CloudWatch dashboard.	February 15, 2017
Added metrics for Amazon Polly	Added metrics for Amazon Polly.	December 1, 2016
Added metrics for Amazon Managed Service for Apache Flink	Added metrics for Amazon Managed Service for Apache Flink.	December 1, 2016
Added support for percentile statistics	You can specify any percentile, using up to two decimal places (for example, p95.45). For more information, see Percentiles.	November 17, 2016
Added metrics for Amazon Simple Email Service	Added metrics for Amazon Simple Email Service.	November 2, 2016
Updated metrics retention	Amazon CloudWatch now retains metrics data for 15 months instead of 14 days.	November 1, 2016
Updated metrics console interface	The CloudWatch console is updated with improvements to existing functionality and new functionality.	November 1, 2016
Added metrics for Amazon Elastic Transcoder	Added metrics for Amazon Elastic Transcoder.	September 20, 2016

Change	Description	Release date
Added metrics for Amazon API Gateway	Added metrics for Amazon API Gateway.	September 9, 2016
Added metrics for AWS Key Management Service	Added metrics for AWS Key Management Service.	September 9, 2016
Added metrics for the new Application Load Balancers supported by Elastic Load Balancing	Added metrics for Application Load Balancers.	August 11, 2016
Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2	Added new NetworkPacketsIn and NetworkPacketsOut metrics for Amazon EC2.	March 23, 2016
Added new metrics for Amazon EC2 Spot fleet	Added new metrics for Amazon EC2 Spot fleet.	March 21, 2016
Added new CloudWatch Logs metrics	Added new CloudWatch Logs metrics.	March 10, 2016

Change	Description	Release date
Added Amazon OpenSearch Service and AWS WAF metrics and dimensions	Added Amazon OpenSearch Service and AWS WAF metrics and dimensions.	October 14, 2015
Added support for CloudWatch dashboards	Dashboards are customizable home pages in the CloudWatch console that you can use to monitor your resources in a single view, even those that are spread out across different Regions. For more information, see Using Amazon CloudWatch dashboards.	October 8, 2015
Added AWS Lambda metrics and dimensions	Added AWS Lambda metrics and dimensions.	September 4, 2015
Added Amazon Elastic Container Service metrics and dimensions	Added Amazon Elastic Container Service metrics and dimensions.	August 17, 2015
Added Amazon Simple Storage Service metrics and dimensions	Added Amazon Simple Storage Service metrics and dimensions.	July 26, 2015
New feature: Reboot alarm action	Added the reboot alarm action and new IAM role for use with alarm actions. For more informati on, see Stop, terminate, reboot, or recover an EC2 instance.	July 23, 2015
Added Amazon WorkSpaces metrics and dimensions	Added Amazon WorkSpaces metrics and dimensions.	April 30, 2015

Change	Description	Release date
Added Amazon Machine Learning metrics and dimensions	Added Amazon Machine Learning metrics and dimensions.	April 9, 2015
New feature: Amazon EC2 instance recovery alarm actions	Updated alarm actions to include new EC2 instance recovery action. For more information, see Stop, terminate, reboot, or recover an EC2 instance.	March 12, 2015
Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions	Added Amazon CloudFront and Amazon CloudSearch metrics and dimensions.	March 6, 2015
Added Amazon Simple Workflow Service metrics and dimensions	Added Amazon Simple Workflow Service metrics and dimensions.	May 9, 2014
Updated guide to add support for AWS CloudTrail	Added a new topic to explain how you can use AWS CloudTrail to log activity in Amazon CloudWatch. For more information, see Logging Amazon CloudWatch API and console operations with AWS CloudTrail.	April 30, 2014

Change	Description	Release date
Updated guide to use the new AWS Command Line Interface (AWS CLI)	The AWS CLI is a cross-service CLI with a simplified installation, unified configuration, and consistent command line syntax. The AWS CLI is supported on Linux/Unix, Windows, and Mac. The CLI examples in this guide have been updated to use the new AWS CLI. For information about how to install and configure the new AWS CLI, see Getting Set Up with the AWS CLI Interface in the *AWS Command Line Interface User Guide*.	February 21, 2014
Added Amazon Redshift and AWS OpsWorks metrics and dimensions	Added Amazon Redshift and AWS OpsWorks metrics and dimensions.	July 16, 2013
Added Amazon Route 53 metrics and dimensions	Added Amazon Route 53 metrics and dimensions.	June 26, 2013
New feature: Amazon CloudWatch Alarm Actions	Added a new section to document Amazon CloudWatch alarm actions, which you can use to stop or terminate an Amazon Elastic Compute Cloud instance. For more information, see Stop, terminate, reboot, or recover an EC2 instance.	January 8, 2013
Updated EBS metrics	Updated the EBS metrics to include two new metrics for Provisioned IOPS volumes.	November 20, 2012

Change	Description	Release date
New billing alerts	You can now monitor your AWS charges using Amazon CloudWatch metrics and create alarms to notify you when you have exceeded the specified threshold. For more information, see Create a billing alarm to monitor your estimated AWS charges.	May 10, 2012
New metrics	You can now access six new Elastic Load Balancing metrics that provide counts of various HTTP response codes.	October 19, 2011
New feature	You can now access metrics from Amazon EMR.	June 30, 2011
New feature	You can now access metrics from Amazon Simple Notification Service and Amazon Simple Queue Service.	July 14, 2011
New Feature	Added information about using the `PutMetricData` API to publish custom metrics. For more information, see Publish custom metrics (PutMetricData / EMF).	May 10, 2011
Updated metrics retention	Amazon CloudWatch now retains the history of an alarm for two weeks rather than six weeks. With this change, the retention period for alarms matches the retention period for metrics data.	April 7, 2011
New feature	Added ability to send Amazon Simple Notification Service or Auto Scaling notifications when a metric has crossed a threshold. For more information, see Alarms.	December 2, 2010
New feature	A number of CloudWatch actions now include the MaxRecords and NextToken parameters, which enable you to control pages of results to display.	December 2, 2010

Change	Description	Release date
New feature	This service now integrates with AWS Identity and Access Management (IAM).	December 2, 2010